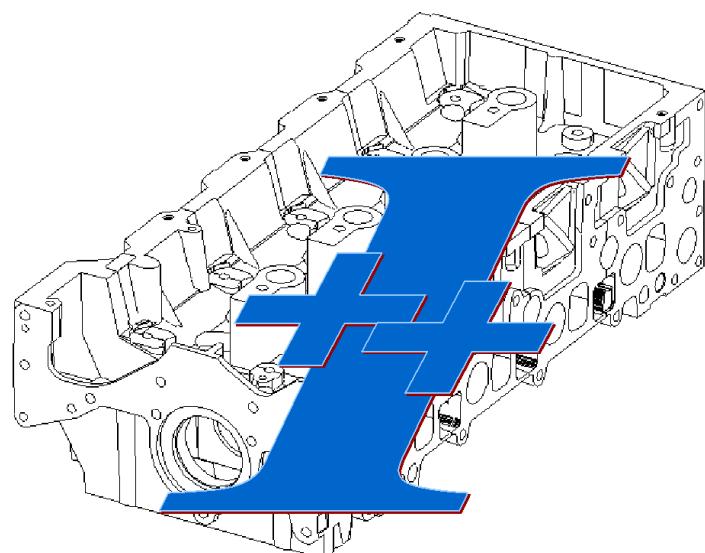




DAIMLERCHRYSLER

VOLVO



DME – Interface

Release 1.09

Contents:

1 I++ WORKING GROUP INFORMATION.....	7
1.1 This specification was created with the assistance of.....	7
1.2 The goal	7
1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment) ..	7
1.4 Requirement.....	7
1.5 What is the intention of the specification ?	7
1.6 Schedule steps	9
1.7 History	9
2 PHYSICAL SYSTEM LAYOUT	10
2.1 DME-Interface Implementations.....	11
2.2 DME-Interface Model.....	11
2.3 Logical System Layout.....	12
2.4 DME-Interface and Subsystems.....	13
2.4.1 Application	13
2.4.2 Monitor	13
2.4.3 Diagnostics	13
2.4.4 Info.....	13
3 HIERARCHY OF COMMUNICATION	14
3.1 Layers	14
3.2 Examples of basic use cases	15
3.2.1 Sequence Diagram: StartSession, EndSession	15
3.2.2 Sequence Diagram: Standard Queue Communication	15
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events)	16
3.2.4 Sequence Diagram: Handling of Unsolicited Errors	17
4 EVENTS	18
4.1 Transaction events, syntax	18
4.2 One shot events	19
4.3 Multiple shot events	19
4.4 Server events	19

5 OBJECT MODEL	20
5.1 Explanation	20
5.2 Reduced Object Model.....	21
5.3 Full Object Model.....	22
5.4 Packaging for visualization, see header files in subdirectories chapter 9	23
5.5 Contents of server.....	23
5.6 Contents of dme.....	24
5.7 Contents of cartcmm.....	24
5.8 Contents of cartcmmwithrotarytable	25
5.9 Contents of toolchanger	26
5.10 Contents of lib and unspecified	27
6 PROTOCOL.....	28
6.1 Communication	28
6.1.1 Character set	28
6.2 Protocol Basics.....	29
6.2.1 Tags.....	29
6.2.2 General line layout.....	30
6.2.2.1 CommandLine	30
6.2.2.2 ResponseLine	30
6.2.2.3 Definitions	30
6.2.3 Transactions.....	31
6.2.3.1 Example.....	31
6.2.4 Events	32
6.2.4.1 Examples	32
6.2.5 Errors	33
6.3 Method Syntax.....	34
6.3.1 Server Methods	34
6.3.1.1 StartSession()	34
6.3.1.2 EndSession().....	34
6.3.1.3 StopDaemon()	34
6.3.1.4 StopAllDaemons().....	35
6.3.1.5 AbortE().....	35
6.3.1.6 GetErrorInfo()	35
6.3.1.7 ClearAllErrors().....	35
6.3.1.8 Information for handling properties	37
6.3.1.9 GetProp()	37
6.3.1.10 GetPropE()	37
6.3.1.11 SetProp().....	37
6.3.1.12 EnumProp()	37

6.3.1.13	EnumAllProp()	38
6.3.2	DME Methods	39
6.3.2.1	Home().....	39
6.3.2.2	IsHomed().....	39
6.3.2.3	EnableUser().....	39
6.3.2.4	DisableUser().....	39
6.3.2.5	IsUserEnabled().....	40
6.3.2.6	OnPtMeasReport().....	40
6.3.2.7	OnMoveReportE().....	40
6.3.2.8	GetMachineClass().....	40
6.3.2.9	GetErrStatusE()	41
6.3.2.10	GetXtdErrStatus().....	41
6.3.2.11	Get()	41
6.3.2.12	GoTo().....	42
6.3.2.13	PtMeas(), PtMeasIJK().....	42
6.3.2.14	Information for Tool Handling.....	44
6.3.2.15	Tool().....	44
6.3.2.16	FindTool()	44
6.3.2.17	FoundTool()	44
6.3.2.18	ChangeTool()	45
6.3.2.19	SetTool().....	45
6.3.2.20	AlignTool().....	45
6.3.2.21	GoToPar().....	46
6.3.2.22	PtMeasPar().....	46
6.3.3	CartCMM Methods.....	47
6.3.3.1	SetCoordSystem()	47
6.3.3.2	GetCoordSystem().....	48
6.3.3.3	GetCsyTransformation()	48
6.3.3.4	SetCsyTransformation(..)	48
6.3.3.5	X()	49
6.3.3.6	Y()	49
6.3.3.7	Z().....	49
6.3.3.8	IJK().....	49
6.3.3.9	X(..)	49
6.3.3.10	Y(..)	50
6.3.3.11	Z(..)	50
6.3.3.12	IJK(..)	50
6.3.4	ToolChanger Methods	51
6.3.5	Tool Methods (Instance of class KTool)	51
6.3.5.1	GoToPar().....	51
6.3.5.2	PtMeasPar().....	51
6.3.5.3	ReQualify().....	51
6.3.6	GoToPar Block	51
6.3.7	PtMeasPar Block	52

7 ADDITIONAL DIALOG EXAMPLES 53

7.1 StartSession..... 53

7.2 Move 1 axis..... 53

7.3	Probe 1 axis	53
7.4	Move more axis in workpiece coordinate system	54
7.5	Probe with more axis.....	54
7.6	Set property.....	54
7.7	Get, read property	55
8	ERROR HANDLING	56
8.1	Classification of Errors.....	56
F1:	Error severity classification.....	56
8.2	List of I++ predefined errors.....	56
9	MISCELLANEOUS INFORMATION.....	58
9.1	Coordination of company related extensions.....	58
9.2	Initialization of TCP/IP protocol-stack	58
9.3	Closing TCP/IP connection	58
9.4	EndSession and StartSession.....	58
9.5	Pre-defined Server events.....	58
9.5.1	KeyPress	58
9.5.2	Clearance or intermediate point set	59
9.5.3	Pick manual point	59
9.5.4	Change Tool request	59
9.5.5	Set property request	59
10	C++ AND HEADER FILES FOR EXPLANATION	61
10.1	\main\main.cpp	61
10.2	\server	61
10.2.1	\server\server.h.....	61
10.2.2	\server\part.h.....	62
10.2.3	\server\server.cpp.....	62
10.3	\dme.....	64
10.3.1	\dem\dme.h	64
10.4	\cartcmm.....	65
10.4.1	\cartcmm\cartcmm.h	65
10.4.2	\cartcmm\eulerw.cpp	66
10.4.3	\cartcmmwithrottbl\cartcmmwithrottbl.h	67
10.5	\toolchanger.....	67

10.5.1	\toolchanger\toolchanger.h	67
10.5.2	\toolchanger\tool.h	69
10.5.3	\toolchanger\toolab.h	70
10.5.4	\toolchanger\toolabc.h	70
10.5.5	\toolchanger\gotoparams.h	71
10.5.6	\toolchanger\ptmeaspars.h	72
10.5.7	\toolchanger\param.h	72
10.6	Most important of lib	73
10.6.1	\lib\axis.h	73
10.6.2	\lib\eulerw.h	74
10.6.3	\lib>tag.h	74
10.6.4	\lib\ipptypedef.h.....	75
10.6.5	\lib\ippbaseclasses.h.....	75

1 I++ Working Group Information

1.1 This specification was created with the assistance of

Hans-Martin Biedenbach,	AUDI AG
Josef Brunner,	BMW
Kai Gläsner,	DaimlerChrysler
Dr. Günter Moritz,	Messtechnik Wetzlar
Jörg Pfeifle,	DaimlerChrysler
Josef Resch,	Zeiss IMT

I++ is a working group of five European Car manufacturers (Audi, BMW, DaimlerChrysler, VW and Volvo).

1.2 The goal

The I++ working group defined a requirement specification with the goal to achieve a new programming system for inspection devices. (not only for CMM's)

This specification will describe the I++ application protocol for the following types of DME's:

- 3D coordinate measuring machines including multiple carriage mode
- Form testers
- Camshaft, crankshaft measuring machines

The spec allows to access all basic DME functionality.

1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)

I++ turn one's attention to the difficulties of the interfaces. So I++ defined a team, who are responsible to workout a requirement specification for a neutral I++ DME interface.

1.4 Requirement

We demand a clear definition, that the DME vendor is responsible for the accuracy of his measurement equipment, in the sense that all necessary functions related to the equipment accuracy have to be implemented in the neutral I++ DME interface.

All calibration data, no matter where created, must be stored in the DME interface.

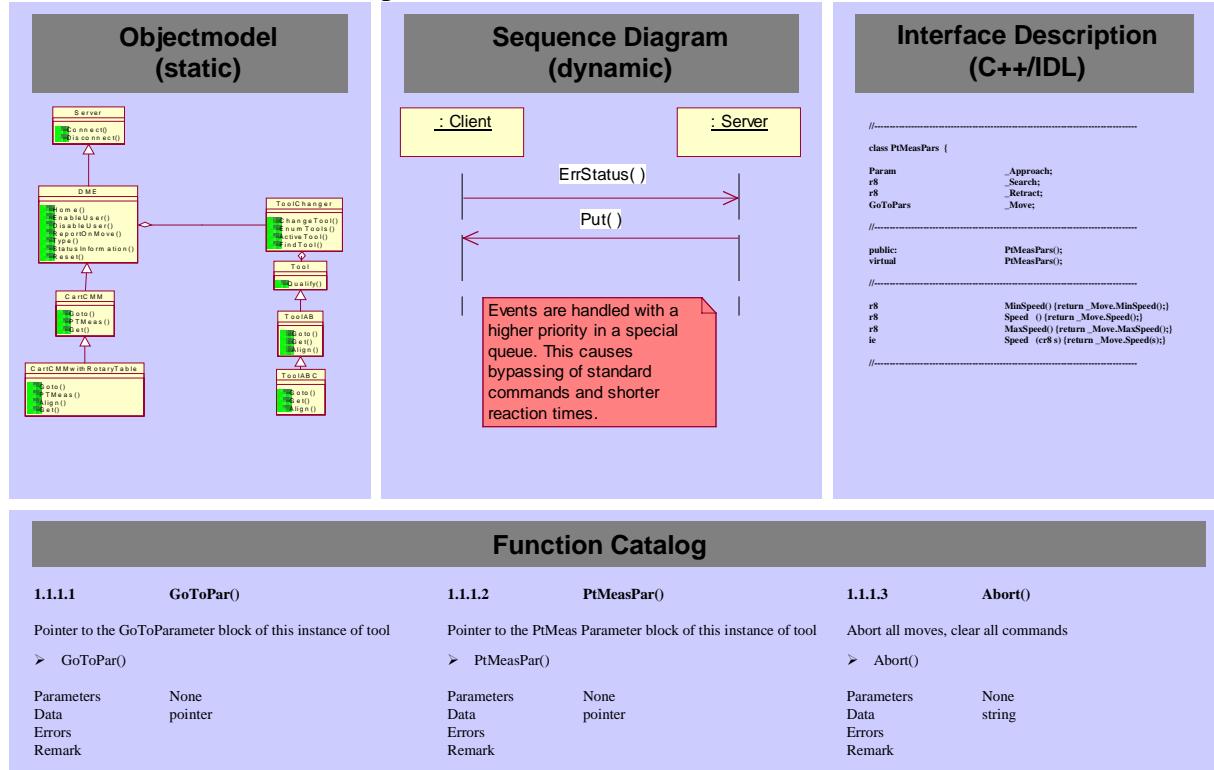
NIST will produce tools for testing the I++ DME Interface specification. These would be made freely availavle outside NIST. Simulated Server/Client for verification, development and certification Scenarios will be provided.

1.5 What is the intention of the specification ?

- To use State-Of-The-Art technology but useable in legacy systems
 - Definition of interface should be independent of transport-layer and transport-technology

- It should provide Scaleability
„an easy machine should have an easy interface“
- Extendability
It should be possible to add new types of machines
- Encapsulation
The complexity and vendor-specific know-how of the real machine should, can be hidden behind the interface
- Self-Explaining, Consistent, Complete
Though being complex the interface should be in a notation, that can be easily understood

Picture 1: Methods for description



The following requirement specification is capable of further developments, this means the specification is valid for CMM's as well as other measurement equipments.

1.6 Schedule steps

Changes from 1.0 to 1.1 :	Multiple arms (port numbers...) Proceeding object model of tools incl. Articulation heads and holders...
Changes from 1.1 to 1.2 :	Scanning, hints, collision handling (predefined parameter blocks),
Changes from 1.2 to 1.3 :	Rotary table
Changes from 1.3 to 1.4 :	Form testers
Changes from 1.4 to 1.5 :	Camshaft, crankshaft measuring machines
Changes from 1.5 to 2.0 :	Optical sensors (based on OSIS requirements)

Unscheduled extension:

- Probe-calibration-parameters-protocol
 - Separate GUI and qualification routines, handle qualification process in client application,
 - List of input-parameters necessary for calibration, all calibration data, no matter where created, must be stored in the DME, for simple probes e.g. indexable touchtrigger-probes PH9-type.
- Add Jog-Box-Display methods
- Use Unicode for strings
- Export tool-assembly information
- New CsY's: JogDisplayCsY, JogMoveCsY, SensorCsY
- Handling more than one socket between client and server

1.7 History

Multiple arms:

Changes: 6.3.3, 6.3.3.3, 6.3.3.4, 10 becomes Appendix A

Added: 10

2 Physical System Layout

This section is intended for helping to explain the context of this specification. It is not part of the specification.

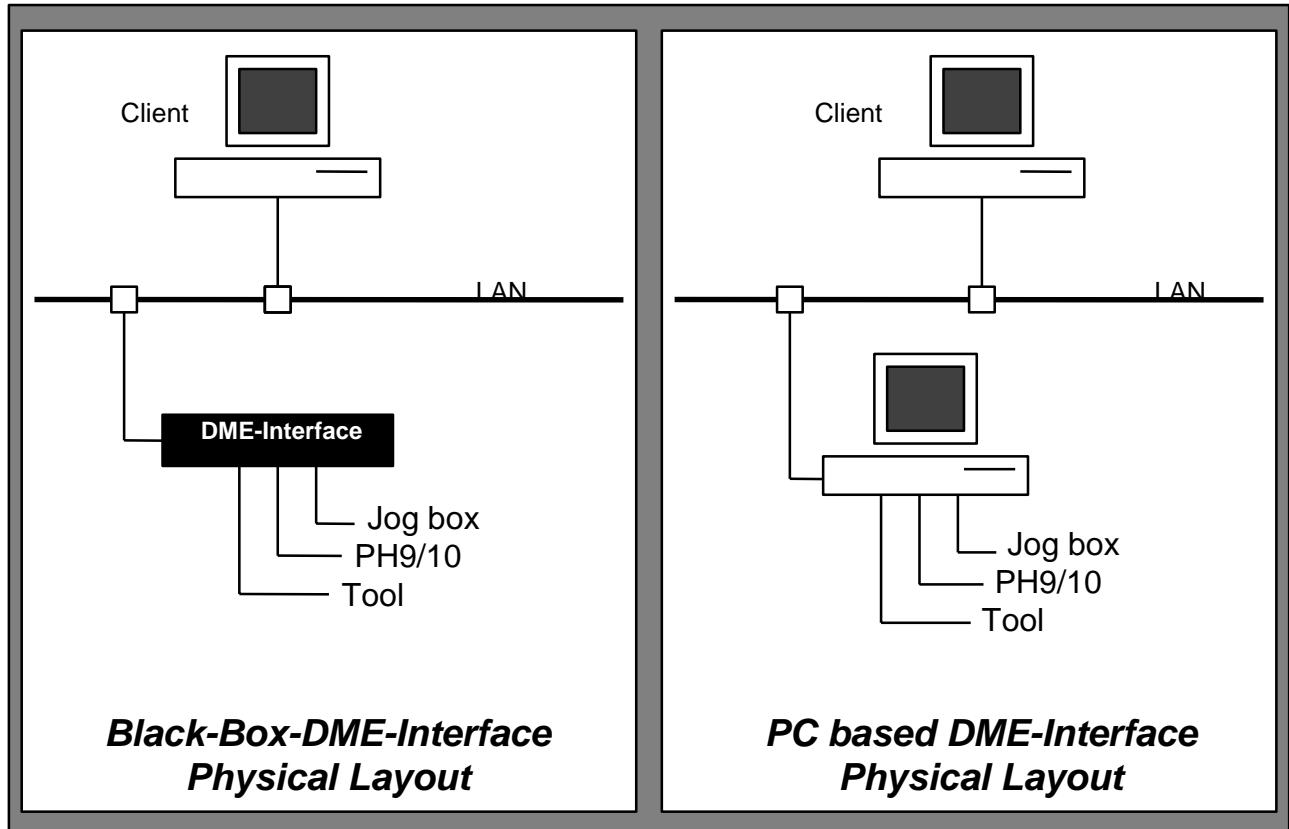
Picture 2 shows two examples of the physical system layout for these types of machines.

In both examples the main components are

Client computer (Client) and

DME-Interface

Machine (including frame, motors, scales, ...)



Client and DME-Interface are connected through a local area network (LAN).

Both client and DME-Interface use TCP/IP sockets for communication.

The client computer runs the application software for the measurement task.

The DME-Interface implements all functionality required to drive the machine.

The application software on the client talks to the DME-Interface in order to execute elementary measurement tasks (picking points, scanning, ...).

This specification describes the protocol that the client uses to run the machine through the DME-Interface.

2.1 DME-Interface Implementations

The main difference between the two implementations of the DME-Interface in Picture 1 is the physical implementation of the DME-Interface, which is

PC based or

“Black Box” based

While PC based DME-Interface provide a direct physical (screen, keyboard) user interface the black box based system provides no direct user interface.

PC based systems may provide additional low-level user interfaces that help the user to control and monitor the machine.

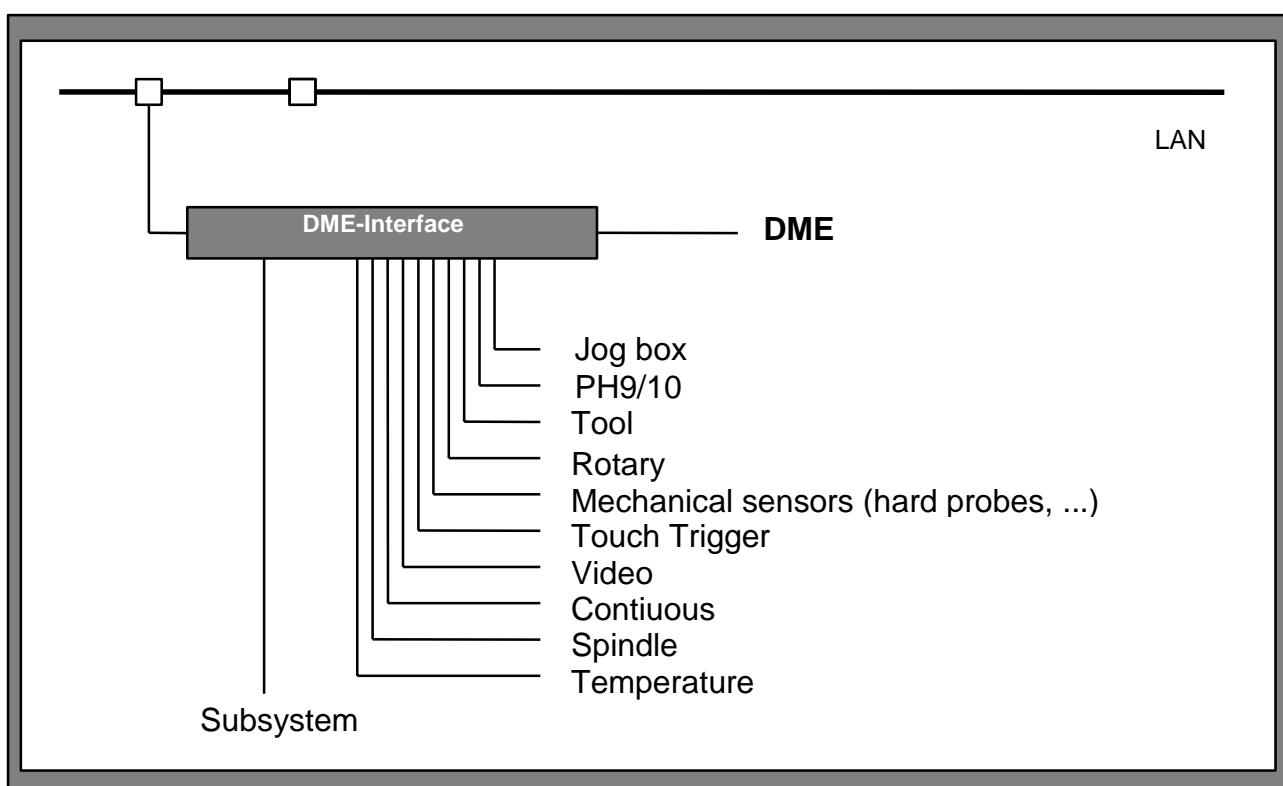
“Black Box” based system have a potential cost advantage.

2.2 DME-Interface Model

Picture 3 shows the system layout we will use in this document for explanations.

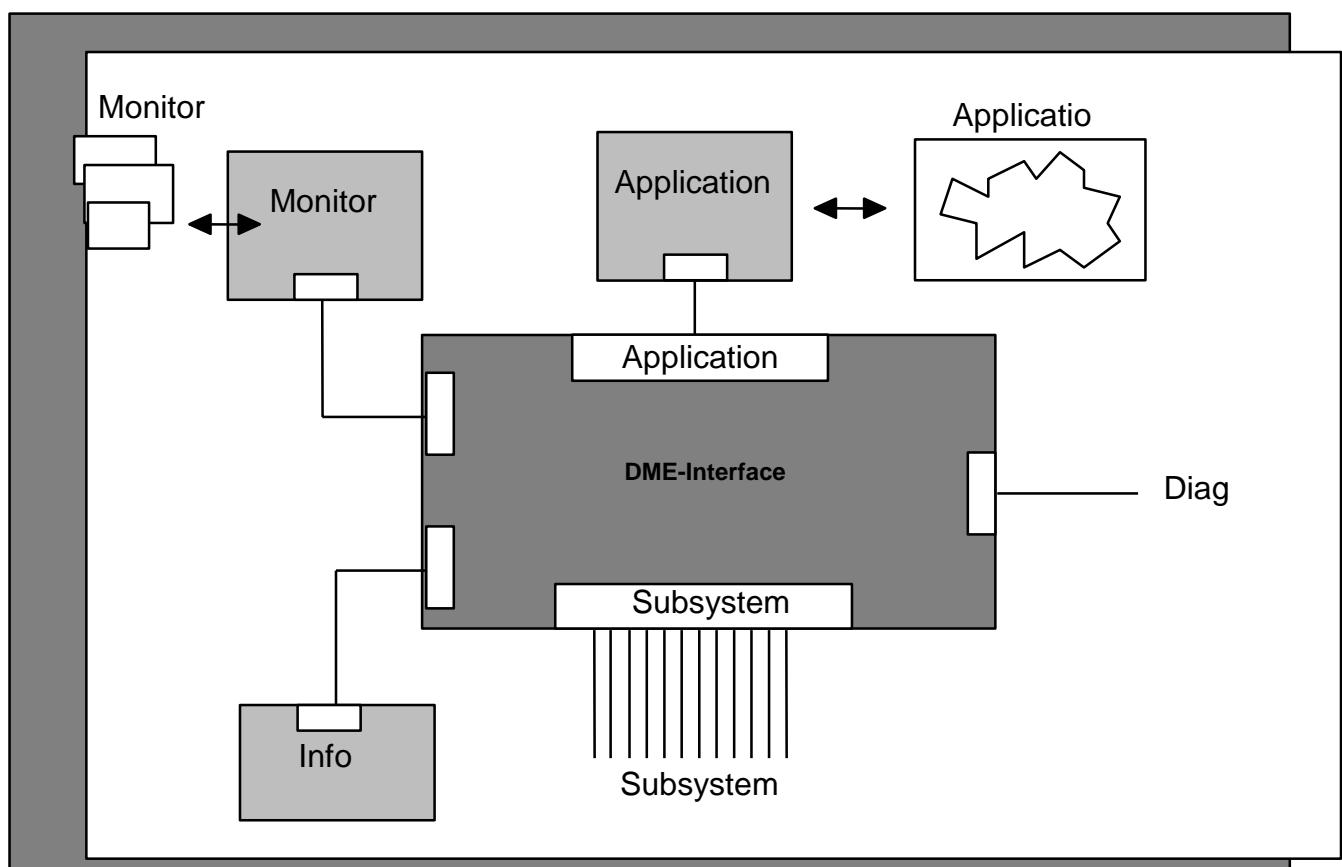
It is important to recognize, that all subsystems are linked to the DME-Interface.

This implies, that the client must use the protocol, to access subsystem functionalities, like rotating a PH10.



2.3 Logical System Layout

Picture 4 shows the logical layout of the system with the following components:
DME-Interface and subsystems
Application
Monitor
Diagnostics



2.4 DME-Interface and Subsystems

The DME-Interface is a PC based or „Black Box“ based piece of hardware that connects to all subsystems (Picture 3).

We will call the piece of software that runs within the “DME-Interface”.

The driver handles all subsystems and provides TCP/IP sockets for communication.

When the DME is powered up, the driver will create up to 4 TCP/IP ports

Application port	(required)	port No. 1294
Monitor port	(optional)	
Diagnostics port	(optional)	
Info port	(optional in V.1.0 will be required in future version)	

2.4.1 Application

The application is a piece of software that runs on the client computer and that uses the application port to run the DME.

This specification describes the protocol used on the application port.

The port number 1294 is internationally defined for this connection.

This port is the only one to start any movements of machine or tool. Only this allows to change any parameter.

2.4.2 Monitor

The machine monitor (monitor) is a piece of software that is used to display controller specific information like current machine position, active probe, ...

It connects to the monitor port to receive the displayed information from the DME-Interface.

The monitor is an optional component.

The controller may implement an equivalent functionality, for examples by displaying the machine position on the jog box display.

In most cases the DME vendor will supply the monitor.

A description of the monitor is not part of this specification.

2.4.3 Diagnostics

The machine diagnostics (diagnostics) is a piece of software that is used to display diagnostic information necessary to service, repair or setup the DME.

It connects to the diagnostic port to receive information from the DME-Interface.

The diagnostic is an optional component.

The DME vendor supplies the diagnostics.

A description of the diagnostics is not part of this specification.

2.4.4 Info

The info is a piece of software that runs on the client computer and that uses the info port to provide information about the machine parameters (axis, sensors,...)

This specification describes the protocol used on the info port.

The functions possible on the info-port are a subset of the functions possible on the application-port. **On this port machine moving commands and setting of parameters are prohibited. Only information receiving dialog is allowed.**

3 Hierarchy of Communication

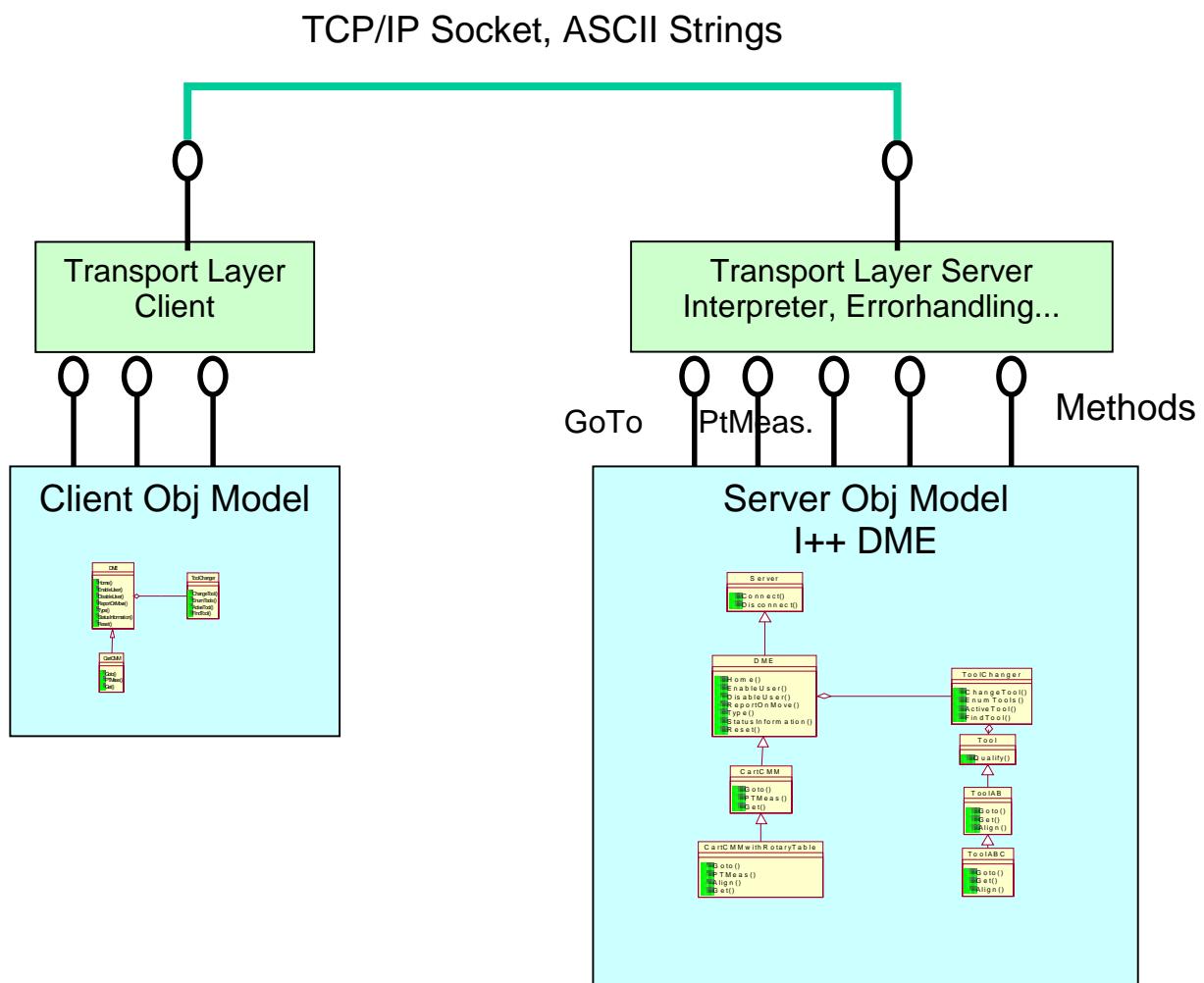
3.1 Layers

The properties of the measuring equipment and the methods to handle them are defined by the object model, see picture 4.

The actual defined transport layer is to transmit ASCII strings via TCP/IP socket.

The layers are separated to have the chance to change the transport layer to future technologies.

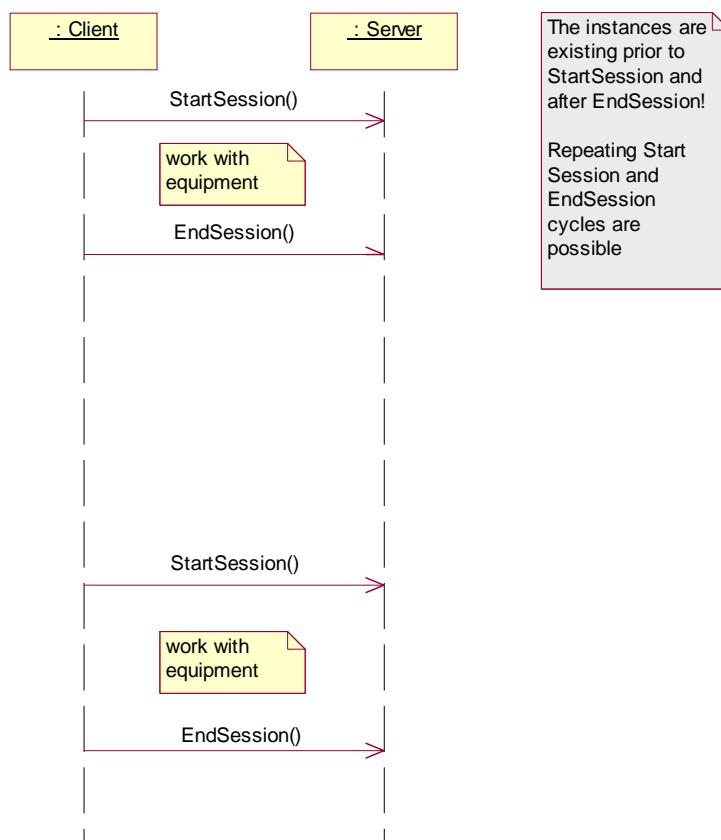
Picture 5



3.2 Examples of basic use cases

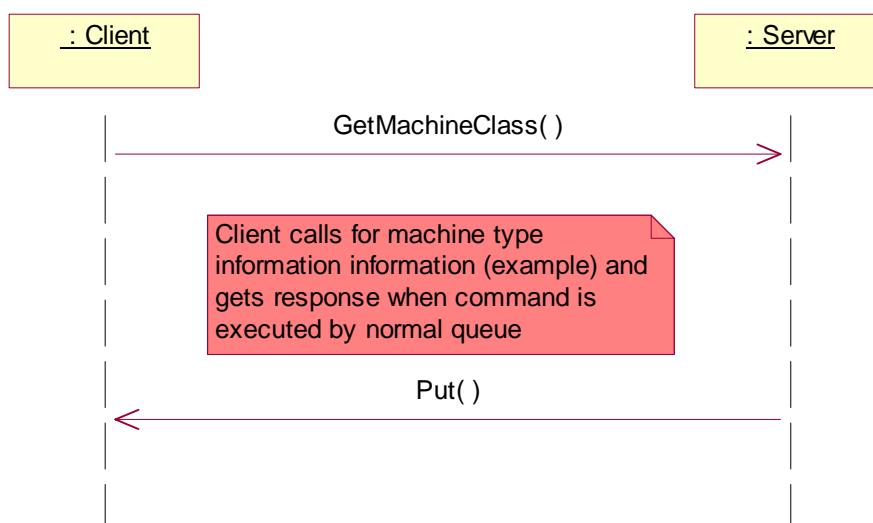
3.2.1 Sequence Diagram: StartSession, EndSession

Picture 6



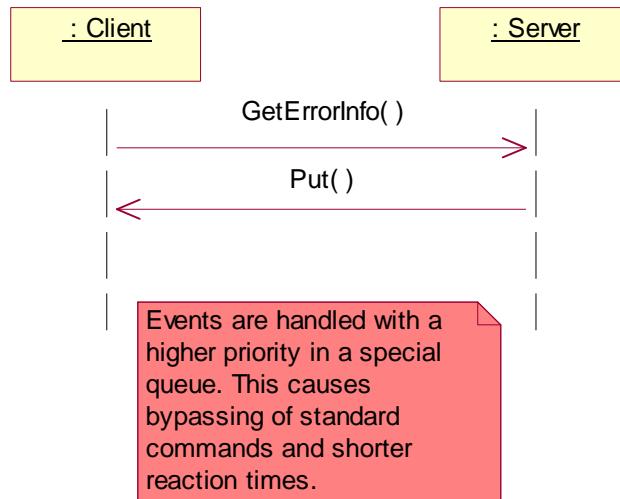
3.2.2 Sequence Diagram: Standard Queue Communication

Picture 7



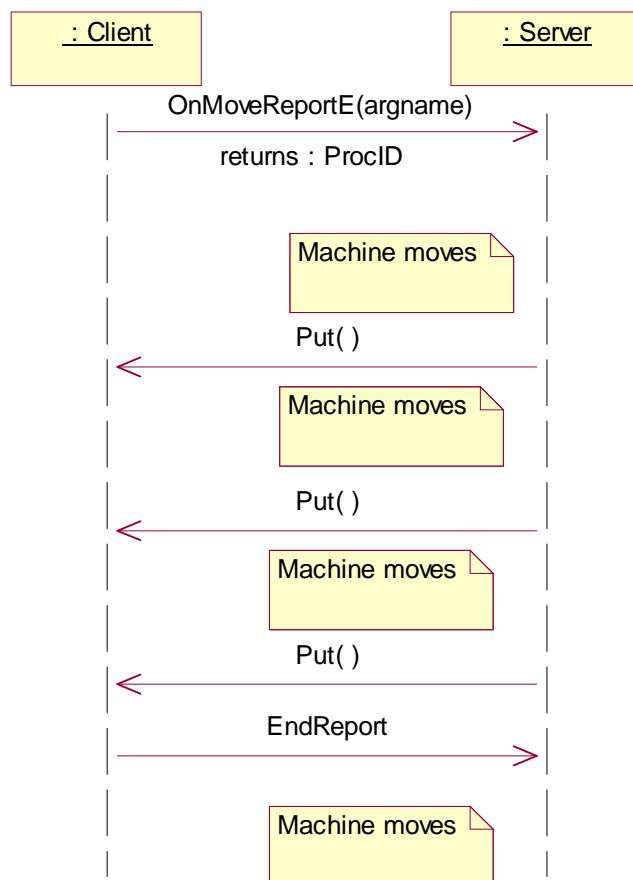
Sequence Diagram: Event, Fast Queue Communication (Single Shot Events)

Picture 8



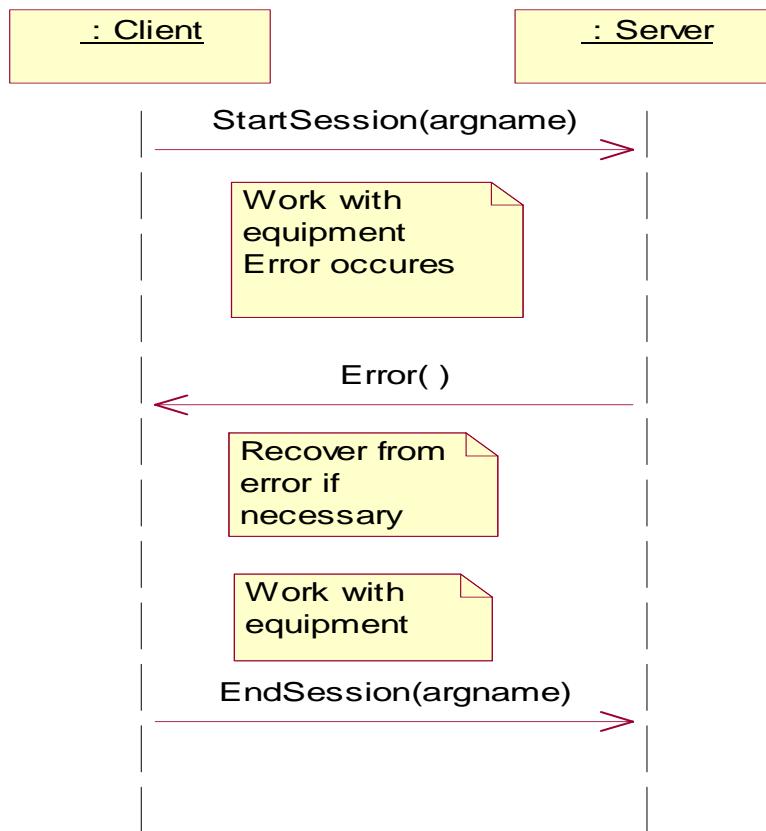
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events)

Picture 9



3.2.4 Sequence Diagram: Handling of Unsolicited Errors

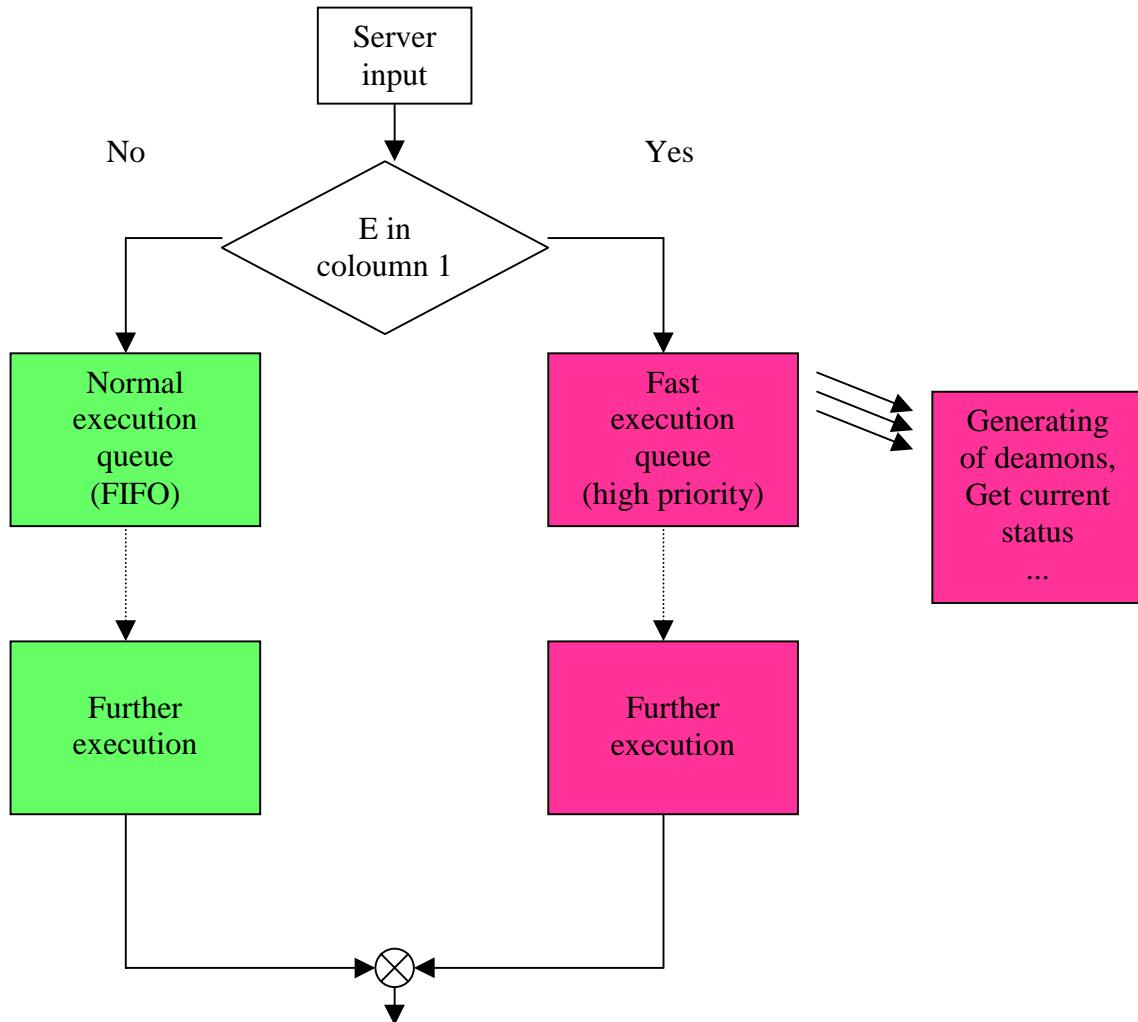
Picture 10



4 Events

To increase performance and to reduce traffic on the interface the Event transactions are created. Events use tags starting with E.

Picture 11: Explanation of the difference between normal and fast queue.
See also sequence diagramm chapter 6.2.4



4.1 Transaction events, syntax

Event transactions are initiated by the client.

Event requests are handled by the server with a higher priority as the synchronous communication. This means that the requests can bypass the normal command queue in the server.

In addition to normal transaction processing, the server will trigger an event. Legal tags are tags starting with E0001 up to E9999. The tag E0000 is reserved for events with no relation to legal tags.

4.2 One shot events

These Events are used to generate exactly one asynchronous reaction of the server. F.I. getting asynchronous status or position information.

The transaction creates a daemon that triggers an event. The daemon will die after firing the event.

4.3 Multiple shot events

The transaction creates a daemon that triggers events based on a condition. The client must stop this daemons explicitly by a StopDeamon („Event transaction tag“) method.

4.4 Server events

Server events use tag E0000. They are used to report manual hits, key strokes, supported machine status changes...

5 Object Model

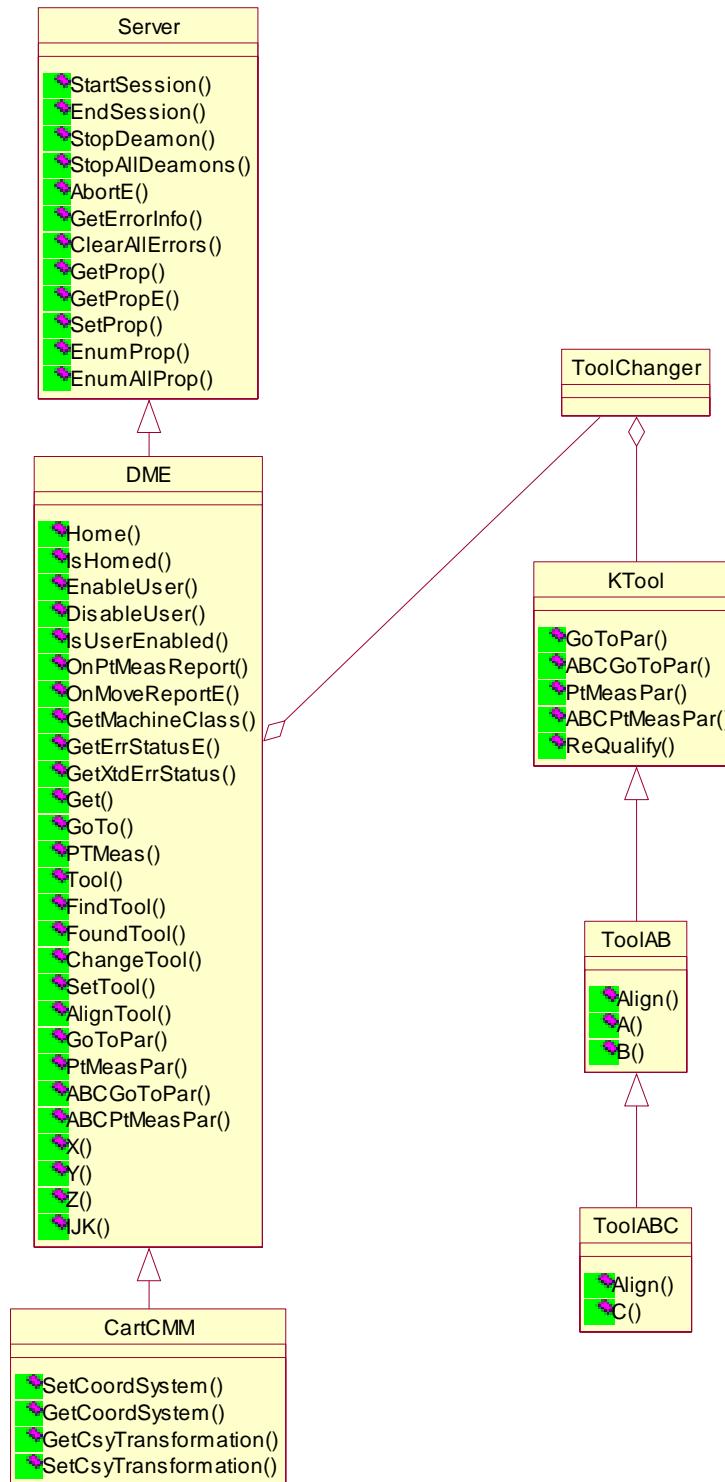
5.1 Explanation

The following diagrams (picture 12 and 13) show the designed class structure of the interface. It shows

- the classes representing the main components of a real coordinate measurement equipment (in this, first case coordinate measurement machine)
- the organization of methods and properties in this classes
- the relations between the main classes, the generalizations (specialization vice versa), the aggregations...
- this object model defines the structure of the interface and the syntax. It defines how to set and get the properties of the virtual components of this machine (chapter 6)
- Picture 11 is generated to help at a first step with the most important commands.
- Picture 12 is reengineered from and consistent to the header files (chapter 9). It shows also programming aspects as virtual definitions of methods in upper classes as DME and also property aspects as GoToPar and PtMeasPar blocks.

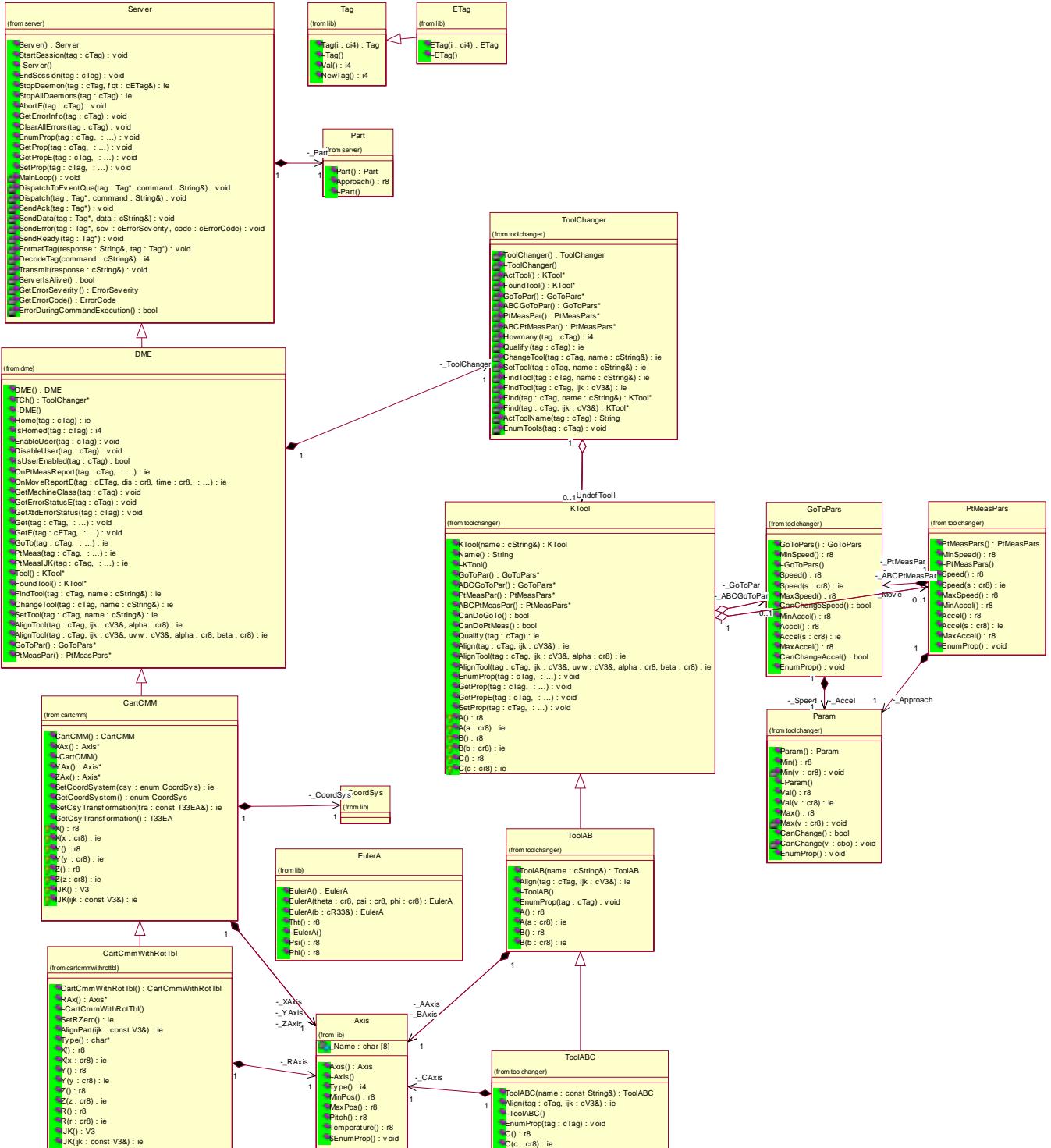
5.2 Reduced Object Model

Picture 12, basics, outside view, method oriented



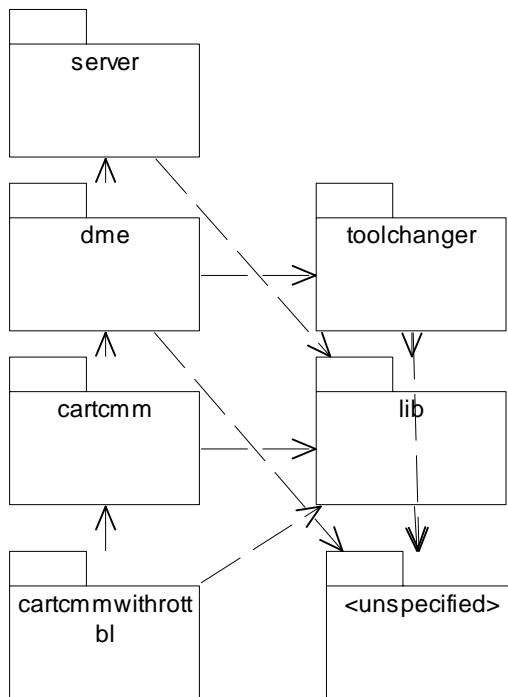
5.3 Full Object Model

Picture 13, please zoom the .pdf file view.



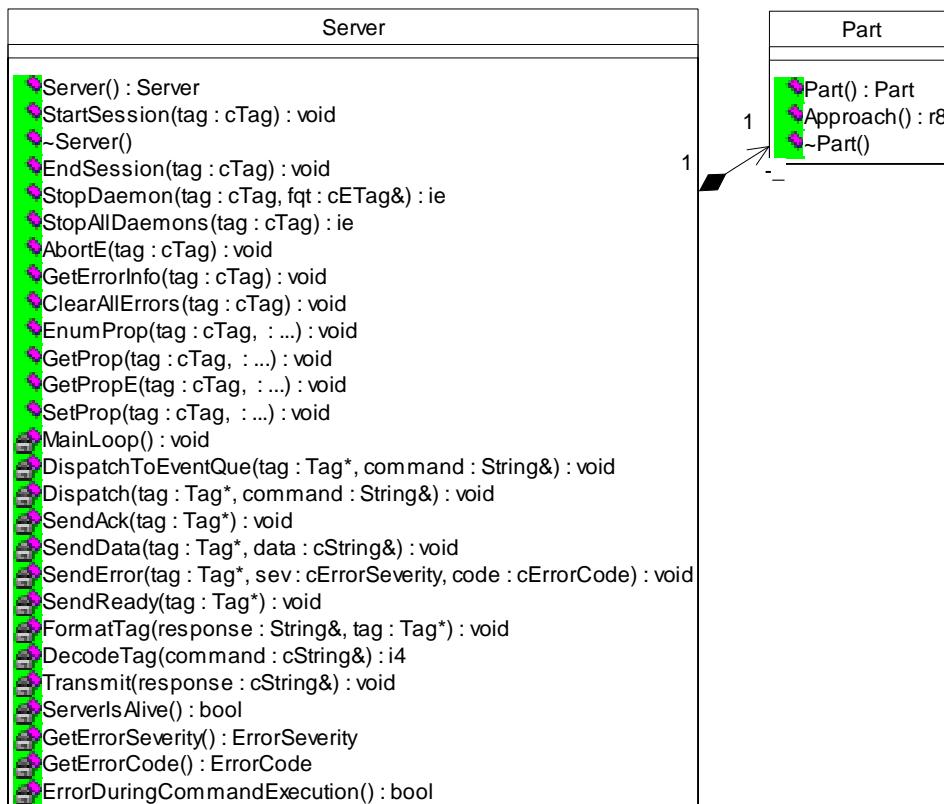
5.4 Packaging for visualization, see header files in subdirectories chapter 9

Picture 14



5.5 Contents of server

Picture 15



5.6 Contents of dme

Picture 16

DME
• DME() : DME • TCh() : ToolChanger* • ~DME() • Home(tag : cTag) : ie • IsHomed(tag : cTag) : i4 • EnableUser(tag : cTag) : void • DisableUser(tag : cTag) : void • IsUserEnabled(tag : cTag) : bool • OnPtMeasReport(tag : cTag, : ...) : ie • OnMoveReportE(tag : cETag, dis: cr8, time : cr8, : ...) : ie • GetMachineClass(tag : cTag) : void • GetErrorStatusE(tag : cTag) : void • GetXtdErrorStatus(tag : cTag) : void • Get(tag : cTag, : ...) : void • GetE(tag : cETag, : ...) : void • GoTo(tag : cTag, : ...) : ie • PtMeas(tag : cTag, : ...) : ie • PtMeasJK(tag : cTag, : ...) : ie • Tool() : KTool* • FoundTool() : KTool* • FindTool(tag : cTag, name : cString&) : ie • ChangeTool(tag : cTag, name : cString&) : ie • SetTool(tag : cTag, name : cString&) : ie • AlignTool(tag : cTag, ijk: cv3&, alpha : cr8) : ie • AlignTool(tag : cTag, ijk: cv3&, uvw : cv3&, alpha : cr8, beta : cr8) : ie • GoToPar() : GoToPars* • PtMeasPar() : PtMeasPars*

5.7 Contents of cartcmm

Picture 17

CartCMM
• CartCMM() : CartCMM • XAx() : Axis* • ~CartCMM() • YAx() : Axis* • ZAx() : Axis* • SetCoordSystem(csy : enum CoordSys) : ie • GetCoordSystem() : enum CoordSys • SetCsyTransformation(tra : const T33EA&) : ie • GetCsyTransformation() : T33EA • X() : r8 • X(x : cr8) : ie • Y() : r8 • Y(y : cr8) : ie • Z() : r8 • Z(z : cr8) : ie • IJK() : V3 • IJK(ijk : const V3&) : ie

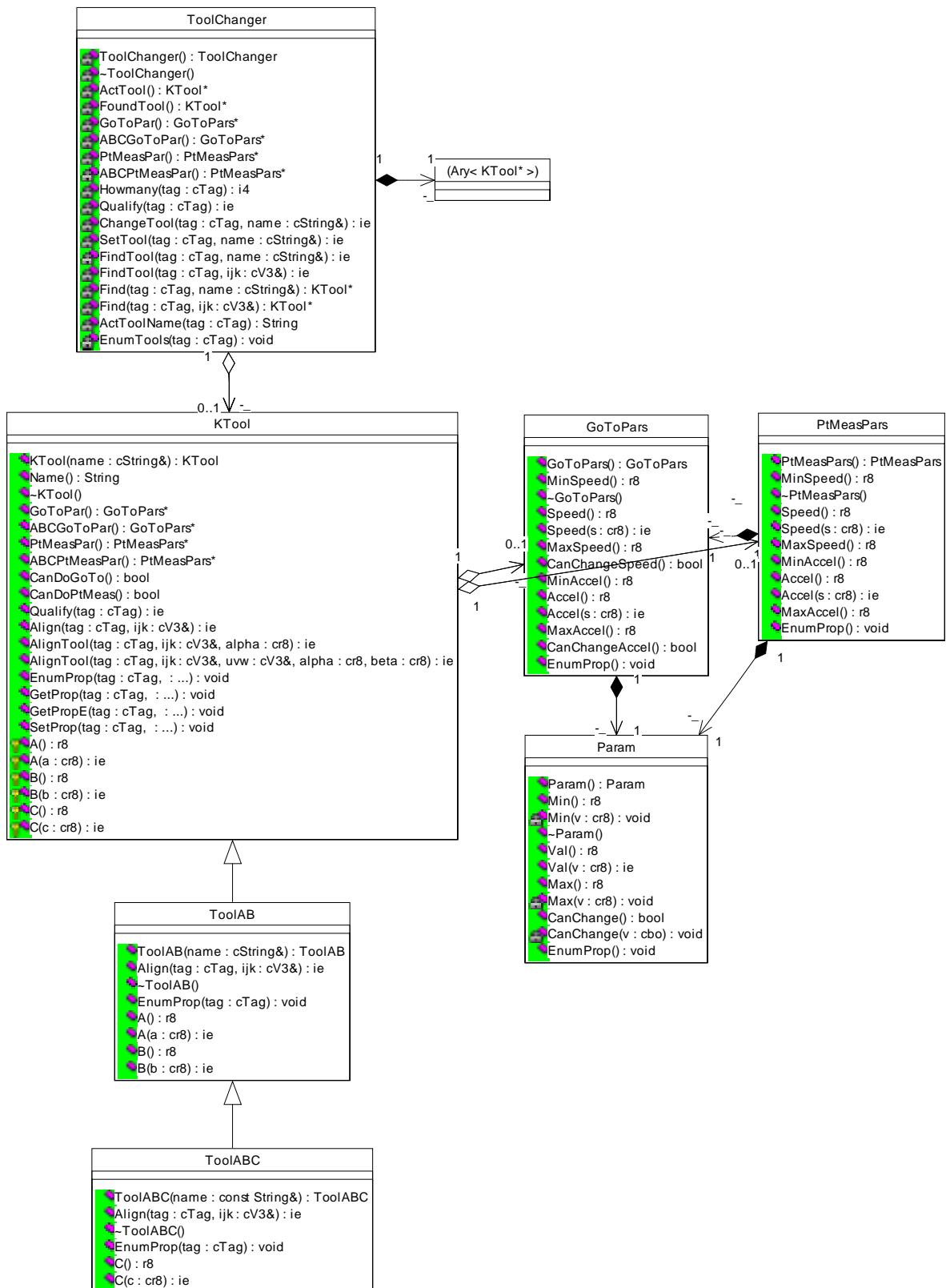
5.8 Contents of cartcmmwithrotarytable

Picture 18

CartCmmWithRotTbl
CartCmmWithRotTbl() : CartCmmWithRotTbl
RAx() : Axis*
-CartCmmWithRotTbl()
SetRZero() : ie
AlignPart(ijk : const V3&) : ie
Type() : char*
X() : r8
X(x : cr8) : ie
Y() : r8
Y(y : cr8) : ie
Z() : r8
Z(z : cr8) : ie
R() : r8
R(r : cr8) : ie
IJK() : V3
IJK(ijk : const V3&) : ie

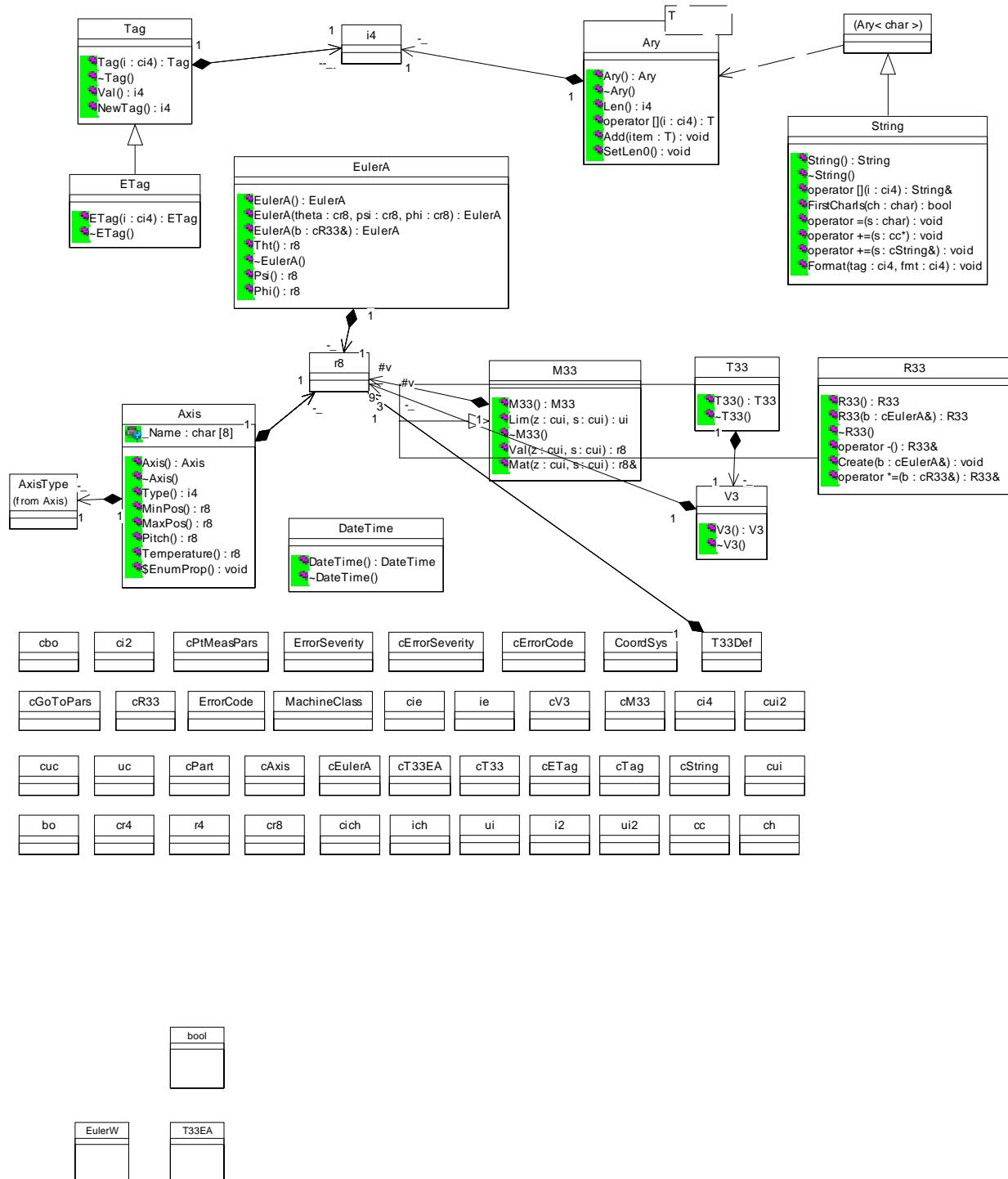
5.9 Contents of toolchanger

Picture 19



5.10 Contents of lib and unspecified

Picture 20



6 Protocol

6.1 Communication

Communication between application client and I++DME server is based on the standard TCP/IP protocol. It uses the application-port with port-no. 1294.

6.1.1 Character set

All bytes received and send on the application port of the server are interpreted as 7 bit ASCII characters.

Only characters in the range from ASCII code = 32 (space) to ASCII code = 126 (~) can be used.

In addition the character pair Carriage Return (<CR>, ASCII code = 13) and Line Feed (<LF>, ASCII code = 10) is used as line terminator

<CR> and <LF> must always be send as a pair in the order <CR> followed by <LF>.

Any character sent outside of this range will cause a “Fatal ServerError”.

Units

Dimensions: mm

Time: sec

Angles: decimal degrees

Temperatur: degrees Celsius

Force: N

String formats

All strings are enclosed in double-quotes (“”). Double-quotes in strings are forbidden. String-length is limited to 255 characters, quotes not included.

Number formats

Decimal-point is used. Allowed formats are: Scientific notation (character for exponent e or E), fixed format with maximum 16 valid digits.

6.2 Protocol Basics

- The protocol is line oriented.
- Each line must be terminated by <CR><LF>.
- The maximum number of characters in a line should not exceed 65535.
- A line send from the client to the server is called CommandLine.
- A line send from the server to the client is called ResponseLine.

In examples the terminating <CR><LF> is not shown !

6.2.1 Tags

The first 5 characters of each CommandLine represent a tag.

The client generates these tags. The client uses two types of tags:

- CommandTag
- EventTag

A CommandTag is a 5 digit decimal number with leading zeros present.

The number must be between 00001 and 09999.

The client is responsible for incrementing the tag number each time it sends a command.

Examples for tag created by the client:

```
04711      // tag is ok
01710      // ok
00020      // ok
20          // error; only 2 digits
00000      // error; out of range must be >=1 and <=9999
```

An EventTag is a 4 digit decimal number that is preceded by the character E(ASCII code=69).

The number must be between 0001 and 9999.

The client must use the same tag counter for CommandTag and EventTag.

To differ in the command layer also between the normal and fast queue, commands for the fast queue end with an upper case E. The reason is to be independent from the transport layer.

The client is responsible for incrementing the tag counter each time it sends a command.

Examples for tag created by the client:

```
E3333      // tag is ok
E0456      // ok
E0000      // error; out of range must be >=1 and <=9999
E20        // error; only 3 characters
A4711      // error; illegal first character
```

As for a CommandLine the first 5 characters of a ResponseLine represent a tag (ResponseTag).

During normal command processing by the server it will use the tag received from the client as ResponseTag so the client can use this tag to relate the ResponseLine to a CommandLine.

In addition the server can send a ResponseLine using ResponseTag E0000 for reporting unsolicited events to the client.

6.2.2 General line layout

From now on we will use

- Command as a synonym for CommandLine
- Response as a synonym for ResponseLine.

6.2.2.1 CommandLine

The first 5 characters in each command line represent the CommandTag.

Character at column 6 must be a space (ASCII code = 32).

The command starts at column 7.

6.2.2.2 ResponseLine

The first 5 characters in each Response line represent the ResponseTag.

Character at column 6 must be a space (ASCII code = 32).

Character at column 7 must be one of the following:

- &
- %
- #
- !

The meaning of character at column 7 is explained later.

Character at column 8 must be a space when the line length is greater than 7.

Example:

00004 # X(99.93), Y(17.148)

6.2.2.3 Definitions

In the following we will use

- Ack as a synonym for a ResponseLine where the 7th character is a &
- Transaction complete
 - as a synonym for a ResponseLine where the 7th character is a %
- Data as a synonym for a ResponseLine where the 7th character is a #
- Error as a synonym for a ResponseLine where the 7th character is a !

6.2.3 Transactions

The basic protocol unit is a transaction. For each transaction, the client will create a tag. The tag identifies the transaction.

- Transactions are initiated by the client.
- The same tag is used during a transaction.
- Transaction can overlap, which means, that a client can start a new transaction after having received an Ack from the server.
- When using overlapped transaction, tags send to the server must be unique.
- When using overlapped transactions and the server is too busy to accept new transactions it must delay sending the Ack until it is ready to accept a new transaction.
- When using overlapped transaction, the server must make sure, that the Ack, Data (Error) and Transaction complete are send back to the client in the right order. This means, if transaction 00001 is started before transaction 00002 the server is not allowed to send a Data, Error or Transaction complete from transaction 00002 before the Transaction complete from transaction 00001. At any point in time the server is allowed to send a line starting with an EventTag.

A transaction is complete after the server sends the Transaction complete. If a transaction is complete, all processing on the driver side related to the transaction has completed.

6.2.3.1 Example

Client to Server	Server to Client	Comment
00001 Home()		Use tag 00001 for home command, client sends „home” command
	00001 &	Server accepts command (Ack)
	00001 %	Server reports transaction complete
00002 GoTo(X(100))		Move to x=100
	00002 &	command accepted (Ack)
	00002 %	position reached (Transaction complete)
00003 GoTo(X(100000))		moving out of limits
	00003 &	command accepted
	00003 ! Error(2, 1, GoTo, Machine Limit Encountered)	Error message
	00003 %	transaction complete
00004 Get(X(), Y())		get position of x, y axis
	00004 &	
	00004 # X(99.93), Y(17.148)	x and y position

	00004 %	Transaction complete
--	---------	----------------------

6.2.4 Events

At any point in time the server may notify that something happened by sending an event to the client.

If the event is triggered by a transaction, the tag used is that of the transaction.

The server must first send an Ack before it can send the Response with the EventTag.

This Response can then be send before or after the transaction complete.

If a event is triggered by a command, f.I. ErrStatusE, the server handles the execution of the command (responding of th error status) with a higher priority. The transaction complete is responded in the order of the standard queue.

At any point in time the server can send a Response with EventTag E0000 to inform the client that something unsolicited has happened in the server.

6.2.4.1 Examples

Unsolicited error message

Client to Server	Server to Client	Comment
	E0000 ! Error(3, 9, HealthCheck, Emergency button activated)	A unsolicited error message occurs
		In this example the server must display error and inform user what to do

Assume the user moves the machine using joysticks and the server wants to report this movement

Client to Server	Server to Client	Comment
00048 EnableUser()		
	00048 &	
	00048 %	
E0553 OnMoveReportE(Time(1),dis(20 ,X()Y()Z())		
	E0553 &	
	E0553 %	
		Now the user moves the machine
	E0553 # X(50), Y(433), Z(500)	
	E0553 # X(50), Y(433), Z(520)	
	...	

		Now the client wants to stop reporting of the server and sends
00049 StopDaemon(E0553)		
	00049 &	
	E0553 # X(50), Y(433), Z(530)	
	00049 %	no events with tag E0553 may follow

6.2.5 Errors

If the server detects an error condition it will report the error using the tag of the Command it was executing when the error was detected.

The server will abort all pending transactions.

The client must invoke the ClearAllErrors() method before the server can continue processing commands.

6.3 Method Syntax

The reference for this description is the C++ class definition that is part of this documentation. Please note, that in the class description the first argument of all methods is Tag. This argument is converted into a CommandTag or EventTag as described before and therefore not part of this documentation (see Server::FormatTag() method).

6.3.1 Server Methods

A session defines the time period after the client has sent a StartSession() until the client sends a EndSession() to the server.

Several states are preserved when the server is shut down, e.g. the active tool.

6.3.1.1 StartSession()

After having completed the connection between client and server on TCP/IP level (see chapter 9.2) the StartSession method initiates the connection between client and server. The server can be sure that the client will invoke StartSession() only once during a session.

- StartSession()

Parameters None

Data None.

Error None.

Remarks The server may for example use this method to perform initial checks
 Like which tool is active, ...

The method does not perform any initializations, which means that the server is in a state that was left after power up or in a state that was left over from the previous session. The client can be sure, that no events or daemons are pending from the session before.

6.3.1.2 EndSession()

The client invokes this method to end a session between client and server.

The client must close the TCP/IP connection (see chapter 9.3) after the transaction is complete.

- EndSession()

Parameters The method has no parameters.

Data None.

Error No errors are returned.

Remarks The method must make sure, that all daemons are stopped and no events are send after it completes. The following states of the server are preserved upon connection of the next client:

Active tool

Active coordinate system

6.3.1.3 StopDaemon()

The client invokes this method to stop a daemon identified by its EventTag.

- StopDaemon(EventTag)

Parameters EventTag of daemon to be stopped

Data None.

Error 0513: Daemon Does Not Exist

6.3.1.4 StopAllDaemons()

The client invokes this method to stop all daemons.

- StopAllDaemons()

Parameters None

Data None.

Error None

Remarks The method must make sure, that all daemons are stopped and no events are send after it completes.

6.3.1.5 AbortE()

The client invokes this method to abort all pending transactions and if possible the current one.

- AbortE()

Parameters None

Data None.

Error None

Remarks The client must invoke the ClearAllErrors() method before the server will process new methods.

6.3.1.6 GetErrorInfo()

The client invokes this method to retrieve the error-information stored in the server

- GetErrorInfo()

Parameters Error-Number

Data Strings

Error None

6.3.1.7 ClearAllErrors()

The client invokes this method to enable the server to recover from an error

- ClearAllErrors()

Parameters None

Data None.

Error None

Examples

Client to Server	Server to Client	Comment
00051 GoTo(X(1000)		
	00051 &	
00052 GoTo(Y(300)		
	00052 &	
		The client wants to abort the moves and sends
00053 AbortE()		
	00053 &	
	00051 ! Error(..”Transaction Aborted”)	
	00051 %	
	00052 ! Error(..”Transaction Aborted”)	
	00052 %	
	00053 %	
		If the client now sends
00054 Get(X())		
	00054 &	
	00054 ! Error(..Error Processing Method)	
	00054 ! Error(..Use Clear All Errors To Continue)	
	00054 %	the server will still be in a error state
00055 ClearAllErrors()		
	00055 &	
	00055 %	the server is now ready to accept new method calls
00056 Get(X())		
	00056 &	
	00056 # X(23)	
	00056 %	

6.3.1.8 Information for handling properties

Each object of the system has to provide functionality to support the following functions

- GetProp(), GetPropE()
- SetProp()
- EnumProp(), EnumAllProp()

6.3.1.9 GetProp()

The client uses this methods to query properties of the system.

- GetProp()

Parameters An enumeration of one or more of the following methods can be argument.

Tool.PtMeasPar.Speed()
Tool.GoToPar.Accel()
or other methods that return properties.

Data The format is defined by the method enumerated.

Errors Errors of the enumerated methods.

6.3.1.10 GetPropE()

The client uses this methods to query the position of the active tool. GetPropE is handled with a high priority. See GetProp().

6.3.1.11 SetProp()

The client uses this methods to set properties of the system.

- SetProp()

Parameters An enumeration of one or more of the following methods can be argument.

Tool.PtMeasPar.Speed(100)
Tool.GoToPar.Accel(10)
or other methods that set properties.

Data The format is defined by the method enumerated.

Errors Errors of the enumerated methods.

6.3.1.12 EnumProp()

The client uses this methods to query properties of the system.

- EnumProp()

Parameters A pointer to a object, e.g. parameter block.

	Tool.PtMeasPar() Tool.GoToPar()
Data	Returns the names of all values The client can use the type information provided to check, if the returned name is a value or a property. The property type is returned "Number" "String" "Property" ! Means class which has own properties See example 7.7.
Errors	Errors of the enumerated methods.

6.3.1.13 EnumAllProp()

The client uses this methods to query properties of the system.

- EnumAllProp()

Parameters A pointer to a object, e.g. parameter block.

	Tool()
Data	Returns the names of all values and the names of all child properties of the property. The client can use the type information provided to check, if the returned name is a value or a property. The property type is returned "Number" "String" "Property" See example 7.7.
Errors	Errors of the enumerated methods.

6.3.2 DME Methods

6.3.2.1 Home()

The client uses this method to home the machine. The server must be homed before the client can invoke methods that move the machine.

➤ Home()

Parameters None.
Data None.
Errors 1005: Error During Home

6.3.2.2 IsHomed()

The client uses this method to query if the machine is homed

➤ IsHomed()

Parameters None
Data IsHomed(Bool).
 Bool = 0 not homed
 Bool = 1 is homed
Errors None.
Remark To get the information of one axis GetProp() functionality should be used.

6.3.2.3 EnableUser()

The client uses this method to enable user interaction with the machine.

➤ EnableUser()

Parameters None.
Data None
Errors None.
Remarks The method will have arguments in the next version to allow the client to enable only a subset of the user interface elements like specific keys or joysticks only.

6.3.2.4 DisableUser()

The client uses this method to disable user interaction with the machine.

➤ DisableUser()

Parameters None.
Data None
Errors None.
Remarks The server calls this method implicitly whenever the client calls a method that physically moves the machine.

6.3.2.5 IsUserEnabled()

The client uses this method to query if the user is enabled.

➤ IsUserEnabled()

Parameters	None.
Data	IsUserEnabled(Bool).
	Bool = 0 user is disabled
	Bool = 1 user is enabled
Errors	None.
Remarks	The client should check if the user is enabled after each StartSession() and not rely on a default.

6.3.2.6 OnPtMeasReport()

The client uses this method to define which information the server should send to the client when the server has completed the PtMeas command.

➤ OnPtMeasReport ()

Parameters	The enumeration may not be empty. See parameters used at command Get() , chapter 6.3.2.11
Data	None.
Errors	0510: Bad Property .
Remarks	The server will send a report after the PtMeas commands has completed.

6.3.2.7 OnMoveReportE()

This is a command for the Fast Queue!

The client uses this method to define which information the server should send to the client while the machine is moving.

➤ OnMoveReportE ()

Parameters	Time(s), Dis(d), ... See parameters used at command Get() , chapter 6.3.2.11 If the enumeration is empty, no reports are send.
Data	None.
Errors	0510: Bad Property .
Remarks	The server will send a report if the time interval s has elapsed or the machine has moved more than d millimeters. The report frequency must not be higher than 10 Hz.
Example	OnMoveReportE(Time(0.5), Dis(0.2), X(), Y(), Z())

6.3.2.8 GetMachineClass()

The client uses this method to query the type of machine.

➤ GetMachineClass()

Parameters None.

Data One of the following must be returned:
GetMachineClass("CartCMM")

Errors None .

6.3.2.9 GetErrStatusE()

This is a command for the Fast Queue!

The client uses this method to query the error status of server.

➤ GetErrStatusE()

Parameters None.

Data ErrStatus(Bool).
 Bool = 1 in error
 Bool = 0 ok

Errors None .

6.3.2.10 GetXtdErrStatus()

The client uses this method to query the extended error status of server.

➤ GetXtdErrStatus()

Parameters None.

Data The server may send one or more lines of status information like

IsHomed(1)
IsUserEnabled(0)

...

Errors as well as one or more Errors like
 1009: Air Pressure Out Of Range
 0512: No Daemons Are Active)

6.3.2.11 Get()

The client uses this method to query the position of the active tool. Also temperatures and calibrated tool properties can be requested.

➤ Get()

Parameters An enumeration of one or more of the following methods can be argument.

X()
Y()
Z()
Tool().A()

or other methods that return axis information. Also temperatures and calibrated tool properties can be requested.

Data	The format is defined by the method enumerated.
Errors	Errors of the enumerated methods.
Remarks	The with “Get” requestable parameters can not be set directly.

6.3.2.12 GoTo()

The client uses this method to perform a multi axis move to the target position using the active tool.

➤ GoTo(..)

Parameters	An enumeration of one or more of the following methods can be argument.
	X(..)
	Y(..)
	Z(..)
Or methods that change tool properties (like Tool.A(), Tool.B()).	

Data	None
Errors	Errors of the enumerated methods.
Implicit	Tool.GoToPar
Remarks	The server will move the machine, so that all axes enumerated will start to move and stop to move at the same time. The tool moves on a straight line in the MachineCsy. Sets implicitly DisableUser().

6.3.2.13 PtMeas(), PtMeasIJK()

The client uses this method to execute a single point measurement using the active tool.

Parameters necessary (Speed, clearance distance, ..) are defined by the active tool

➤ PtMeas(..)

Parameters	An enumeration of one or more of the following methods can be argument.
	X(..)
	Y(..)
	Z(..)
	IJK(..)

Data	As defined by OnPtMeasReport() method
Errors	1006: Surface Not Found.
Implicit	Tool.PtMeasPar
Remarks	Tool.GoToPar The PtMeas() method is processed by the server as follows:

If a IJK vector is present

- The vector I, J, K is normalized

- The position X, Y, Z is shifted into I, J, K direction by the following values:

Part.Clearance() used for the actual approach
(Tool.Approach() used as minimum for warning)
Tool.Radius()
{picture explaining.....}

This new position is called approach position.

- The server moves the machine to the approach position. This move is executed like an implicit GoTo().
- The position X, Y, Z is shifted in the opposite I, J, K direction by the value of Tool.Search(). This position is called end of search position.
- The server moves the machine to end of search position using the PtMeasPar of the Tool().
- If the tool has part contact during this move the server latches the position of the center of the ActTool and reports to the client as defined by OnPtMeasReport().
- After contact the server will move the machine according to the value of Tool.Retract() using Tool.GoToPar for the move. If Tool.Retract() is greater or equal zero, the server will shift the contact position into IJK direction by this value and move the machine to this position. If the Tool.Retract() is less than zero, the server will move back to the approach position as defined before.

If a IJK vector is not present

- The position before invocation of the method is the approach position.
- The values of X, Y, Z define the end of search position.
- The direction from the end of search position to the approach position implicitly define a IJK direction.
- The server moves the machine to end of search position using the Tool.PtMeasPar for speed and acceleration.
- If the tool has part contact during this move the server latches the position of the center of the ActTool and reports to the client as defined by OnPtMeasReport().
- After contact the server will move the machine according to the value of Tool.PtMeasPar.Retract() using Tool.GoToPar. If this value greater or equal zero, the server will shift the contact position into IJK direction by this value and move the machine to this position. If the Tool.PtMeasPar.Retract() is less than zero, the server will move back to the approach position as defined before.

Note that the end of search position may be outside the move limits, but the part surface inside. In this case the server will report success if the surface is still inside or ErrorMoveOutOfLimits. This behaviour differs from the GoTo() method.

Sets implicitly DisableUser().

6.3.2.14 Information for Tool Handling

To handle special tool behaviours the following tools are defined (predefined, reserved names)

BaseTool	! Holds the default DME capabilities, e.g. speed, acceleration
RefTool	! Supports all standard tool properties
NoTool	! Can only move but not measure
UnDefTool	!

6.3.2.15 Tool()

The client uses this method to select a pointer to the actual activated tool. It can also return a pointer to NoTool! See Example 7.6.

➤ Tool()

Parameters	None
Data	None
Errors	

6.3.2.16 FindTool()

Method to get a pointer to a tool with a known name. See Example 7.7.

➤ FindTool("Too1")

Parameters	Name of tool to search
Data	FoundTool
Errors	1502: Tool Not Found

6.3.2.17 FoundTool()

A pointer to a tool with a known name selected by FindTool("xxx"). FoundTool() is only valid after a call to FindTool(). See Example 7.7.

➤ FoundTool()

Parameters	None
Data	None
Errors	
Remark	Pointer can also be NoTool!

6.3.2.18 ChangeTool()

Method to change tool by ProbeChanger or manually

- ChangeTool("Tool2")

Parameters	Name of the tool to activate
Data	None
Errors	1502: Tool Not Found...
Remark	If an error occurs during the execution of ChangeTool(), the client is responsible to ask the server, which tool is active then.

6.3.2.19 SetTool()

Forces the server to assume, a given tool is the active tool.

- SetTool("Tool2")

Parameters	Name of the tool to set
Data	None
Errors	1502: Tool Not Found...
Remark	If an error occurs during the execution of SetTool(), the client is responsible to ask the server, which tool is active then.

The server assumes, active tool is "Tool2"

6.3.2.20 AlignTool()

Forces the tool to orientate according to the given vector(s).

- AlignTool(i1,j1,k1, alpha)
- AlignTool(i1,j1,k1,i2,j2,k2, alpha, beta)

Parameters One normalized vector (i1, j1, k1). This vector is anti parallel to the main axis of the tool (away from the surface).

Two normalized vectors (i1, j1, k1, i2, j2, k2). The first vector is anti parallel to the main axis of the tool (away from the surface). The second vector describes the orientation in the working plane.

Maximal allowed error angles (alpha, beta) in which the found orientation may differ from the desired one. In case the angle differs, ToolNotFound is returned. In case alpha or beta are zero no error check is performed.

Data	Returns vectors (same number as set) which describe the reached alignment
Errors	1502: Tool Not Found

Remark If an error occurs during the execution of SetTool(), the client is responsible to ask the server, which tool is active then.

6.3.2.21 GoToPar()

Pointer to the GoToParameter block of the DME

➤ GoToPar()

Parameters None

Data pointer

Errors

Remark

6.3.2.22 PtMeasPar()

Pointer to the PtMeasParameter block of the DME

➤ PtMeasPar()

Parameters None

Data pointer

Errors

Remark

6.3.3 CartCMM Methods

Each CartCMM implements a cartesian machine coordinate system.

Based on this coordinate system the following depend on it:

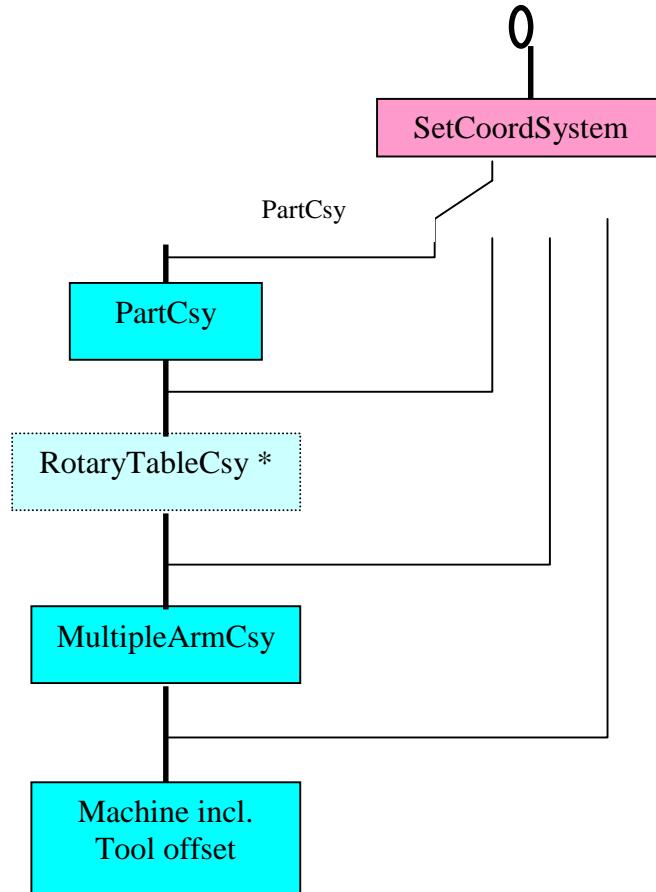
- MachineCsy
- MoveableMachineCsy
- MultipleArmCsy
- PartCsy

The CMM will for some commands use execution parameters that are defined in the context of a tool like speed and acceleration for executing a GoTo command.

Picture XX: Transformation chain model

The multiplearm transformation is implemented also on bridge type or single arm machines

* The RotaryTablCsy is here listed because of consistency reasons of the transformation chain.



6.3.3.1 SetCoordSystem()

The client uses this method to select the coordinate system it wants to work with.

➤ SetCoordSystem(Param)

Parameters One of the following:

MachineCsy
MoveableMachineCsy
MultipleArmCsy
PartCsy

Data None.

Errors 0509: Bad Parameter

6.3.3.2 GetCoordSystem()

The client uses this method to query the server which coordinate system is selected..

➤ GetCoordSystem()

Parameters None.

Data CoordSystem(Arg)

Arg can be one of the following:
MachineCsy
MoveableMachineCsy
MultipleArmCsy
PartCsy

Errors None.

6.3.3.3 GetCsyTransformation()

The client uses this method to get the enumerated coordinate transformation back from the server.

➤ GetCsyTransformation(Enumerator)

Parameters Enumerator: PartCsy
JogDisplayCsy
JogMoveCsy
SensorCsy
MultipleArmCsy

Data GetCsyTransformation(X0, Y0, Z0, Theta, Psi, Phi)

Errors None.

Remarks Definition of relation between transformation matrix and parameters is given in C++ class definition. See Section 10.4.2.

6.3.3.4 SetCsyTransformation(..)

The client uses this method to replace the enumerated coordinate transformation.

➤ SetCsyTransformation(Enumerator, X0,Y0,Z0, Theta, Psi, Phi)

Parameters Enumerator: PartCsy
JogDisplayCsy
JogMoveCsy
SensorCsy

MultipleArmCsy

X0,Y0, Z0 define the zero point of the machine coordinate system in part coordinates. Theta, Psi and Phi are Euler angles that define the rotation matrix of the transformation.

Data	None
Errors	1007: Theta Out Of Range
Remarks	See Section 10.4.2.

6.3.3.5 X()

➤ X()

Parameters	None.
Data	X(x)
Errors	1503: Tool Not Defined 0509: Bad Parameter
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.6 Y()

➤ Y()

Parameters	None.
Data	Y(y)
Errors	1503: Tool Not Defined 0509: Bad Parameter
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.7 Z()

➤ Z()

Parameters	None.
Data	Z(z)
Errors	1503: Tool Not Defined 0509: Bad Parameter
Remarks	This method can only be invoked as argument of a Get or OnReport method.

6.3.3.8 IJK()

➤ IJK()

Parameters	None.
Data	IJK(I,j,k)
Errors	0508: Bad Context
Remarks	i,j,k define a direction vector in the actual coordinate system. The vector is not necessarily normalized. Its values are tool dependent. If the client normalizes the vector it should point out of the part material. This method can only be invoked as an argument of OnPtMeasReport().

6.3.3.9 X(..)

➤ X(x)

Parameters	target x position
Data	None. ...
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.10 Y(..)

➤ Y(y)

Parameters	target y position
Data	None. ...
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.11 Z(..)

➤ Z(z)

Parameters	target z position
Data	None. ...
Errors	2500: Machine Limit Encountered 2504: Collision 0508: Bad Context
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.12 IJK(..)

➤ IJK(I,j,k)

Parameters	I,j,k define the X,Y,Z values of a vector.
Data	None
Errors	0508: Bad Context 1010: Vector Has No Norm
Remarks	i,j,k define a direction vector in the machine coordinate system. The vector is not necessarily normalized. Before using the vector, the server must normalize it. This method can only be invoked as an argument of another method.

6.3.4 ToolChanger Methods

Each CMM implements one instance ToolChanger to install and change tools. The methods are available, and described here in the DME-Chapter, because there is exactly one instance.

6.3.5 Tool Methods (Instance of class KTool)

Each CMM implements a class KTool to contain the properties of the tool and the methods to handle them

6.3.5.1 GoToPar()

Pointer to the GoToParameter block of this instance of KTool

➤ GoToPar()

Parameters None
Data pointer
Errors
Remark

6.3.5.2 PtMeasPar()

Pointer to the PtMeasParameter block of this instance of KTool

➤ GoToPar()

Parameters None
Data pointer
Errors
Remark

6.3.5.3 ReQualify()

Method to requalify ActTool

➤ ReQualify()

Parameters None, ActTool is used
Data None
Errors Errormessages during calibration
Remark

6.3.6 GoToPar Block

Each parameter block contains information for

Speed and
Accel.

Each of this physical values are splitted in

Min
Max
Act and
Def.

There is the parameter-block of the active tool accessible via the DME and a parameter-block associated to each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h chapter 10.5.1. The method to access the values are described in the examples 7.7

6.3.7 PtMeasPar Block

Each parameter block contains information for

Speed
Accel
Approach
Search
Retract.

Each of this physical values are splitted in

Min
Max
Act and
Def.

There is the parameter-block of the active tool accessible via the DME and a parameter-block associated to each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h chapter 10.5.1. The method to access the values are described in the examples 7.7

7 Additional Dialog Examples

7.1 StartSession

Client to Server	Server to Client	Comment
		Server and Client must be boot up previously
00001 StartSession		Client connects to server
	00001 &	Server sends acknowledge
	00001 %	Server sends transaction complete

7.2 Move 1 axis

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy, 10, 20, 30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to and from part coordinate system
	00010 &	
	00010 %	
00011 GoTo(X(100))		Move now in part coordinate system
	00011 &	
	00011 %	

7.3 Probe 1 axis

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),To ol.A())		Client defines format for probing result. Valid for every PtMeas command from now on.
	00014 &	
	00014 %	
00015 PtMeas(X(200))		Uses standard method in CartCMM
	00015 &	
	00015 # X(199.998),Y(250.123),Z (300.002),Tool.A(45)	Probing result from server
	00015 %	

7.4 Move more axis in workpiece coordinate system

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy,10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to part coordinate system
	00010 &	
	00010 %	
00011 Goto(X(100),Y(150),Z(200)) 00011 Goto(X(100),Y(150),Z(200),R(180))		Move with more axis Alternatively
	00011 &	
	00011 %	

7.5 Probe with more axis

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),To ol.A())		Valid for every PtMeas command
	00014 &	
	00014 %	
00015 PtMeas(X(200),Y(250),Z(300)) 00015 PtMeas(X(200),Y(250),Z(300),I JK(0,0,1)) 00015 PtMeas(X(200),Y(250),Z(300),I JK(0,0,1),R(180))		Uses standard method in CartCMM Alternatively, with approaching vector Alternatively, with approaching vector and rotary table
	00015 &	
	00015 # X(199.998),Y(250.123),Z (300.002),Tool.A(45)	Result
	00015 %	

7.6 Set property

Client to Server	Server to Client	Comment
00015 SetProp(Tool.PtMeasPar.Speed(100))		Set probing speed of active tool
	00015 &	
	00015 %	

7.7 Get, read property

All properties that are represented as strings are exchanged using double-quotes, e.g. “This is my probe”

Client to Server	Server to Client	Comment
00014 EnumProp(Tool.PtMeasPar())		Get ActTools PtMeas Property list
	00014 &	
	00014 # “Speed”, “Value”	
	00014 # “Accel”, “Value”	
	
	00014 # “Approach”, “Value”	
	00014 %	
00015 GetProp(Tool.PtMeasPar.Speed())		Request for getting probing speed of active tool
	00015 &	
	00015 # Tool.PtMeasPar.Speed(10 0)	
	00015 %	
00016 FindTool(“Probe1”)		Search pointer to Probe 1
	00016 &	
	00016 %	
00017 GetProp(FoundTool.PtMeasPar. Speed())		Get Probing speed of Probe1
	00017 &	
	00017 # Speed(100)	
	00017 %	

8 Error Handling

- Each transaction can generate multiple error messages.
- These messages are headed by the same tag number.

8.1 Classification of Errors

0003 ! Error(F1, F2, F3, Text)

F1: Error severity classification

0: Info

1: Warning

2: Error, client should be able to repair the error

3: Error, user interaction necessary

9: Fatal server error

Errors with classification higher or equal 2 require ClearAllErrors().

F2: Error numbers, 0000-4999, defined by I++ DME

5000-8999 definable from server

9000-9999 definable from client

F3: I++ recommends to serve here the name of the error causing method

8.2 List of I++ predefined errors

Classification in Field F2

0000-0499 Protocol, syntax error

0500-0999 Error generated during execution in DME (see object model)

1000-1499 Error generated during execution in CartCMM... (see object model)

1500-1999 Error generated during execution in ToolChanger (see object model)

2000-2499 Error generated during execution in Tool... (see object model)

2500-2999 Error generated during execution in Axis (see object model)

Defined errors:

Severity class	Error No.	Text
0	0000	Buffer full
2	0001	Illegal tag
2	0002	No space at pos. 5
2	0003	Illegal flag character at pos. 6
2	0004	No space at pos 7
1	0005	Suspend communication
2	0006	Transaction aborted
3	0500	Emergency stop
3	0501	Unsupported command
3	0502	Incorrect parameters
9	0503	Controller communications failure
1	0504	Parameter out of range
3	0505	Parameter not recognized
3	0506	Parameter not supported
3	0507	Illegal command

3	0508	Bad context
3	0509	Bad parameter
3	0510	Bad property
2	0511	Error processing method
1	0512	No daemons are active
2	0513	Daemon does not exist
2	0514	Use ClearAllErrors to continue
3	1000	Machine in error state
2	1001	Illegal touch
9	1002	Axis does not exist
2	1003	No touch
9	1004	Number of angles not supported on current device
2	1005	Error during home
2	1006	Surface not found
3	1007	Theta out of range
3	1008	Target position out of machine volume
3	1009	Air pressure out of range
2	1010	Vector has no norm
3	1500	Failed to re-seat head
3	1501	Probe not armed
3	1502	Tool not found
3	1503	Tool not defined
3	2000	Tool not calibrated
2	2001	Head error excessive force
3	2002	Type of probe does not allow this operation
3	2500	Machine limit encountered [Move Out Of Limits]
3	2501	Axis not active
2	2502	Axis position error
9	2503	Scale read head failure
3	2504	Collision
2	2505	Specified angle out of range

9 Miscellaneous Information

9.1 Coordination of company related extensions

The goal of each company should be to bring extensions to the standard. To allow fast uncoordinated developments try outs before that, specific name spaces are reserved.. The following mechanism is offered:

Commands in Class DME beginning with
BS. Are Brown&Sharpe proprietary
CZ. Are Zeiss proprietary...

The handling of properties is done similary

SetProp(BS()....)
GetProp(BS()...) is B&S proprietary
SetProp(CZ()....)
GetProp(CZ()...) is Zeiss proprietary...
. . .

9.2 Initialization of TCP/IP protocol-stack

After CMM power up the server will create the application port in listen mode.

When the client is started, it will send a connection request to the application port created by the server. The server will confirm the connection and is now ready to work with the client.

9.3 Closing TCP/IP connection

When the client no longer needs the server it will close the connection.

The driver will then listen on the application port for new incoming connection requests.

9.4 EndSession and StartSession

After re-starting a session all previous defined properties are valid again.

9.5 Pre-defined Server events

The following server events are predefined. Please note that all these events are transmitted with tag number E0000 to the client.

9.5.1 KeyPress

Data KeyPress("NameOfKey")

Remarks The server sends this event, if the user is enabled and when the user has pushed button on the jog box.
Key names are to be defined.

9.5.2 Clearance or intermediate point set

Data GoTo(...)

Remarks The server sends this event, if the user is enabled and when the user has pushed the “Clearance Point” button on the jog box.
The GoTo format is defined by OnMoveReport()

9.5.3 Pick manual point

Data PtMeas(...)

Remarks The server sends this event, if the user is enabled and when the user manually picked a point.
The Ptmeas format is defined by OnPtMeasReport()

9.5.4 Change Tool request

Data ChangeTool(“ToolName”)

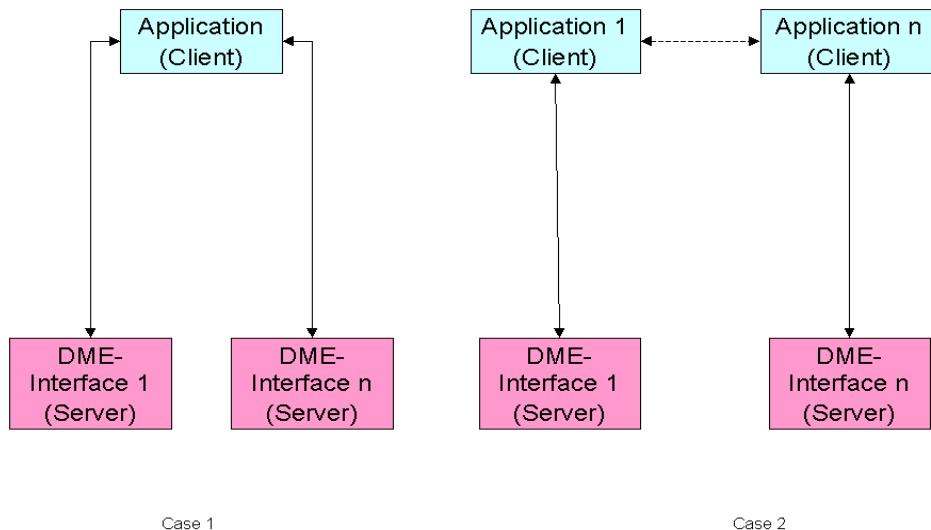
Remarks The server sends this event, if the user is enabled and when the user has pushed a button on the jogbox, that selects a new tool..
This event is only a request. The client has to decide if it should send a ChangeTool() command to execute the change.
Note, that this command does not change the tool.

9.5.5 Set property request

Data SetProp()

Remarks The server sends this event, if the user is enabled and when the user has pushed a button on the jogbox, that changes a property.
This event is only a request. The client has to decide if it should send a SetProp() command to execute the change.
Note, that this command does not change the property.

10 Multiple arm support



- For each column a single I++ DME interface is required.
- The disposition of a feature to be measured on a specific column, the synchronisation of the columns (including collision detection between the columns) and the combination of the results is part of the application task.
- The vendor of the multiple arm system has to provide an application to build the coupling transformation. This can be a stand alone application or an integrated part of the DME interface.
- The following commands are used to set and get these transformations
`SetCsyTransformation(MultipleArmCsy,.....), GetCsyTransformation(MultipleArmCsy)`

The following is a simple example scenario for building the dual arm transformation

- The application connects to I++ DME interface of column 1
- The application measures the artifact used to build the transformation using column 1
- The application disconnects from column 1
- The application connects to DME column 2
- The application measures the artifact used to build the transformation using column 2
- The application disconnects from column 2
- The application calculates the 2 transformations used for column 1 and 2
- The application connects to DME of column 1 and sends the transformation for column 1 using a `SetCsyTransformation(MultipleArmCsy,.....)` command
- The application disconnects from column 1
- The application connects to DME of column 2 and sends the transformation for column 2 using a `SetCsyTransformation(MultipleArmCsy,.....)` command
- The application disconnects from column 2

Appendix A C++ and Header Files for Explanation

1. \main\main.cpp

```
//-----
#include "../cartcmm/cartcmm.h"
//#include "../dme/dme.h"
//#include "../cartcmmwithrottbl/CartCmmWithRotTbl.h"

//-----
//Server      _Server;
//Server*     Srv()    {return &_Server;}

void main () {
    //r8          speed = Srv()->GoToPar()->Speed();
    //ie          r    = Srv()->GoToPar()->Speed(5.0);

//-----
};
```

2. \server

1. \server\server.h

```
#if !defined(AFX_Server_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_Server_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "Part.h"
#include <ETag.h>

//-----

class Server {

    Part      _Part;

//-----

public:      Server  ();
virtual ~Server();

//-----

void          StartSession (cTag tag);                                // connect to client
void          EndSession   (cTag tag);                                // disconnect from
client
ie           StopDaemon  (cTag tag, cETag &fqt);                      // stop daemon
ie           StopAllDaemons(cTag tag);                                // stop all
daemons
void          AbortE       (cTag tag);                                // abort pending
transactions
void          GetErrorInfo (cTag tag);                                // abort pending
transactions

void          ClearAllErrors(cTag tag);

//-----

virtual void  EnumProp     (cTag tag, ...);
virtual void  GetProp      (cTag tag, ...);
virtual void  GetPropE(cTag tag, ...);
virtual void  SetProp      (cTag tag, ...);

//-----

private:
    // these methods are for
        // documentation purpose only
```

```

void      MainLoop          ();
void      DispatchToEventQue (Tag *tag, String &command);
void      Dispatch           (Tag *tag, String &command);
void      SendAck            (Tag *tag);
void      SendData           (Tag *tag, cString &response);
void      SendError          (Tag *tag, cErrorSeverity sev, cErrorCode code);
void      SendReady           (Tag *tag);
void      FormatTag          (String &response, Tag* tag);
i4       DecodeTag          (cString &command);
void      Transmit            (cString &response);

//-----
bool      ServerIsAlive();
ErrorSeverity GetErrorSeverity();
ErrorCode     GetErrorCode();
bool         ErrorDuringCommandExecution();

//-----
};

#endif

```

2. \server\part.h

```

#ifndef !defined(AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include <IppTop.h>

//-----
class Part {

//-----
r8      _Approach;
r8      _XpanCoefficient;
r8      _Temperature;

//-----
public:      Part      ();
virtual     ~Part     ();

//-----
r8      Approach()      {return _Approach;}

//-----
};

#endif

```

3. \server\server.cpp

```

//-----

#include "String.h"
#include "Server.h"

//-----

void      Server::MainLoop()          { // for documentation only ***
    // this method is implemented for
    // documentation purpose only
    String   command;
    Tag*    tag   = Nil;
    do {
        // wait for a command line from client
        // .. Wait(command)
    }
}
```

```

i4          tagval = DecodeTag(command);           // get tag values define by chars from
2 to 5

        if (command.FirstCharIs('E')) {
            tag = new ETag(tagval);
            SendAck(tag);                                // confirmre
        }
    receive
        DispatchToEventQue(tag, command); // do whatever is necessary
    else {
        tag = new Tag(tagval);
        SendAck(tag);                                // confirmre
    }
    receive
        Dispatch(tag, command);                   // do whatever is necessary
    wrong ?
        if (ErrorDuringCommandExecution()) {         // something went
            SendError(tag, GetErrorSeverity(), GetErrorCode()); // send error message
            SendReady(tag);                         // while server is alive
            while (ServerIsAlive());}
    //-----
void    Server::Transmit(cString &response)   {           // for documentation only ***
    // send this string to client
    /*... Send(response) */}

//-----
void    Server::FormatTag(String &response, Tag* tag){ // for documentation only ***
    remove all chars      response.SetLen0();           //
    leading zeros          response.Format(tag->Val(), 5); // format 5 digits with
    if (dynamic_cast<ETag*>(tag) !=Nil) {
        response[1] = 'E';
        response += " ";}                                // use E to indicate event
    // append a space char
//-----
void    Server::SendAck(Tag* tag)   {           // for documentation only ***
    String      response;
    FormatTag(response, tag);
    response += "&";                                // add %
    Transmit(response);}

//-----
void    Server::SendData(Tag *tag, cString &data)   {           // for documentation only ***
    String      response;
    FormatTag(response, tag);
    response += "#";                                // add # and
    space
    response += data;                                // add data
    Transmit(response);}

//-----
void    Server::SendError(Tag *tag, cErrorSeverity sev, cErrorCode code)   {           // for documentation only ***
    String      response;
    FormatTag(response, tag);
    response += "! ";                                // add ! and space
//    response += ...;                                // add error
    Transmit(response);}

//-----
void    Server::SendReady(Tag *tag)                // for documentation only ***
    String      response;

```

```

        FormatTag(response, tag);
        response += "%";
        Transmit(response);
```

//-----

3. \dme

1. \dem\dme.h

```

#ifndef AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_
#define AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../server/Server.h"
#include "../toolchanger/ToolChanger.h"

//-----

class DME : public Server {

    ToolChanger           _ToolChanger;

//-----

    bool                  _IsHomed;
    bool                  _IsUserEnabled;

//-----

public:          DME      ();
virtual         ~DME     ();

//-----

    ToolChanger*          TCh()                      {return &_ToolChanger;}

//-----

    virtual ie             i4          Home (cTag tag)           {}           {return _IsHomed;}
                                IsHomed(cTag tag)

//-----

    virtual void            EnableUser (cTag tag)        {};
    virtual void            DisableUser (cTag tag)       {};
    virtual bool             IsUserEnabled(cTag tag)    {};

//-----

    ie                     OnPtMeasReport (cTag tag, ...);
    ie                     OnMoveReportE (cETag tag, cr8 dis, cr8 time,...);

//-----

    void                  GetMachineClass (cTag tag)    {};
    void                  GetErrorStatusE (cTag tag)   {};
    void                  GetXtdErrorStatus(cTag tag)  {};

//-----

    void                  Get              (cTag tag, ...);
    void                  GetE             (cETag tag, ...);

    ie                   GoTo            (cTag tag,...);
    ie                   PtMeas          (cTag tag,...);
    ie                   PtMeasJK        (cTag tag,...);

//-----

    KTool*                Tool()           const        {return _ToolChanger._ActTool;};
    KTool*                FoundTool()      const        {return _ToolChanger._FoundTool;};

//-----
```

```

ie FindTool (cTag tag, cString &name) {return TCh()->FindTool
(tag, name);}
ie ChangeTool (tag, name); {return TCh()-
>ChangeTool (tag, name);}
ie SetTool (cTag tag, cString &name) {return TCh()->SetTool
(tag, name);}
ie AlignTool (cTag tag, cv3 &ijk, cr8 alpha) {return Tool()->AlignTool (tag, ijk,
alpha);}
ie AlignTool (cTag tag, cv3 &ijk, cv3 &uvw, cr8 alpha, cr8 beta) {return Tool()-
>AlignTool (tag, ijk, uvw, alpha, beta);}

//-----
GoToPars* GoToPar () const {return Tool()->GoToPar();}
PtMeasPars* PtMeasPar () const {return Tool()->PtMeasPar();}

//-----
};

#endif

```

4. \cartcmm

1. \cartcmm\cartcmm.h

```

#ifndef AFX_CartCMM_H_E4F9759D_0A8F_11D3_A3F2_0000F87ABD00_INCLUDED_
#define AFX_CartCMM_H_E4F9759D_0A8F_11D3_A3F2_0000F87ABD00_INCLUDED_

#include "../DME/dme.h"
#include "T33.h"

//-----

class CartCMM :public DME {
    Axis _XAxis;
    Axis _YAxis;
    Axis _ZAxis;

//-----

    CoordSys _CoordSys;
    T33 _PartCoordTrandformation;

//-----

public: CartCMM ();
virtual ~CartCMM () ;

//-----

    Axis* XAx() {return &_XAxis;}
    Axis* YAx() {return &_YAxis;}
    Axis* ZAx() {return &_ZAxis;}

//-----

    virtual ie SetCoordSystem(CoordSys csy);
    CoordSys GetCoordSystem() {return _CoordSys; }

//-----

    ie SetCsyTransformation(const T33EA &tra);
    T33EA GetCsyTransformation();

//-----

protected:

    virtual r8 X(); // return machine position in
    the
    virtual r8 Y(); // selected coordinate
    system
    virtual r8 Z();
    virtual V3 IJK();
```

```

virtual ie X(cr8 x); // move machine to target position
virtual ie Y(cr8 y);
virtual ie Z(cr8 z);
virtual ie IJK(const V3 &ijk);

//-----
};

#endif



## 2. \cartcmm\eulerw.cpp


// EulerA.cpp: implementation of the EulerA class.

#include "R33.h"
#include "EulerW.h"

//-----

cr8 R_Delta = 1e-12;

r8 abs (cr8 x);
r8 sind (cr8 x);
r8 cosd (cr8 x);
r8 Acosd (cr8 x);
r8 Atan2d(cr8 y, cr8 x);

//-----

EulerA::EulerA(cR33 &b){ // create Euler from
    _Psi = 0, // rotation matrix
    _Phi = 0;
    s3 = 0,
    c3 = 0,
    c1 = b.Val(3,3);
    _Theta = Acosd(c1);
    s1 = sind(_Theta);
    if (abs(s1) > R_Delta) { // check if Tht() is 0
        s2 = b.Val(1,3)/s1, // no calculate
        r8 Psi()
        c2 = -b.Val(2,3)/s1;
        _Psi = Atan2d(s2, c2);
    }
    EulerA ew(_Theta, _Psi, _Phi); // use Tht(), Psi(), 0 to create matrix
    R33 r(ew); -r; // and
    calcalate Psi() from orig mat
    R33 rr(b); rr*=r; // and
    matriz build from Tht() Psi()
    r8 c3 = r.Val(1,1);
    r8 s3 = r.Val(2,1);}
    else {
        // Tht()==0, Psi()==0
        c3 = b.Val(1,1);
        s3 = b.Val(2,1);}
        _Phi = Atan2d(s3, c3);} // calculate phi

//-----

void R33::Create(cEulerA &b) {
    r8 psi==0
    c1 = cosd(b.Thet()), // theta==0 &&
    s1 = sind(b.Thet()), // c3
    c2 = cosd(b.Psi()), // -s3
    0
    s2 = sind(b.Psi()), // c3
    0
    c3 = cosd(b.Phi()), // 0
    1
    s3 = sind(b.Phi()); // Mat(1,1) = c2*c3-c1*s2*s3;
    Mat(1,2) = s2*c3+c1*c2*s3;
    Mat(1,3) = s1*s3;
    Mat(2,1) = -c2*s3-c1*s2*c3;
    Mat(2,2) = -s2*s3+c1*c2*c3;
}

```

```

        Mat(2,3) = s1*c3;           //      c2*c3-c1*s2*s3    s2*c3+c1*c2*s3    s1*s3
        Mat(3,1) = s1*s2;           //      -c2*s3-c1*s2*c3   -s2*s3+c1*c2*c3    s1*c3
        Mat(3,2) = -s1*c2;
        Mat(3,3) = c1;             //      s1*s2
        c1

//-----

```

3. \cartcmmwithrottbl\cartcmmwithrottbl.h

```

#ifndef !defined(AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../cartcmm/cartcmm.h"

//-----

class CartCmmWithRotTbl: public CartCMM {

//-----

Axis          _RAxis;

//-----

public:          CartCmmWithRotTbl           ();
virtual ~CartCmmWithRotTbl ();

//-----

Axis*          RAx() {return &_RAxis;}

//-----

ie             SetRZero();
ie             AlignPart(const V3 &ijk);

//-----

virtual char*  Type() {return "CartCMMWithRotTbl";}

//-----

virtual r8     X();                                // return machine position in
virtual r8     Y();                                // selected coordinate
system
virtual r8     Z();
virtual r8     R();
virtual V3    IJK();

virtual ie     X(cr8 x);                          // move machine to target position
virtual ie     Y(cr8 y);
virtual ie     Z(cr8 z);
virtual ie     R(cr8 r);
virtual ie     IJK(const V3 &ijk);

//-----
};

#endif

```

5. \toolchanger

1. \toolchanger\toolchanger.h

```

// ToolChanger.h: interface for the ToolChanger class.

#ifndef !defined(AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112__INCLUDED_)
#define AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112__INCLUDED_

#include "ToolAB.h"

//-----

```

```

class ToolChanger {
    friend class DME;

    KTool* _ActTool;
    KTool* _FoundTool;
    KTool* _DefaultTool;
    KTool* _UndefTool;

    //-----
    Ary<KTool*> _Tools;
    V3             _TransferPosition;
    //-----

    ToolChanger (){

        /*
        String      name      = "DefaultTool"
                    _DefaultTool = new Tool(name);

                    name      = "UndefTool"
                    _UndefTool = new Tool(name);

                    name      = "NoTool"
                    = new Tool(name);
                    _Tool.Add(tool);

                    name      = "ReferenceTool"
                    = new Tool(name);
                    _Tool.Add(tool);

        */
    }

    virtual ~ToolChanger();

    //-----

    KTool* ActTool() const {return _ActTool;}
    KTool* FoundTool() const {return _FoundTool;}
    //-----

    GoToPars* GoToPar() const {GoToPars* r=ActTool()->GoToPar (); if (r==Nil) r=_DefaultTool->GoToPar(); return r;}
    GoToPars* ABCGoToPar()const {GoToPars* r=ActTool()->ABCGoToPar(); if (r==Nil) r=_DefaultTool->ABCGoToPar(); return r;}
    //-----

    PtMeasPars* PtMeasPar() const {PtMeasPars* r=ActTool()->PtMeasPar (); if (r==Nil) r=_DefaultTool->PtMeasPar(); return r;}
    PtMeasPars* ABCPtMeasPar()const {PtMeasPars* r=ActTool()->ABCPtMeasPar(); if (r==Nil) r=_DefaultTool->ABCPtMeasPar(); return r;}
    //-----

    i4 Howmany(cTag tag) {return _Tools.Len();}

    //-----

    ie Qualify (cTag tag) {return _ActTool->Qualify(tag);}
    ie ChangeTool (cTag tag, cString &name);
    ie SetTool (cTag tag, cString &name);
    ie FindTool (cTag tag, cString &name){_FoundTool = Find(tag, name); return (_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
    ie FindTool (cTag tag, cv3 &ijk) {_FoundTool = Find(tag, ijk); return (_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
    String ActToolName(cTag tag) {return _ActTool->Name();}

    //-----

    void EnumTools(cTag tag) {
        name;
        for (i4 i=0; i < _Tools.Len(); i++) {

```

```

        name = _Tools[i]->Name();
        /*send name to client*/}

//-----
private:
KTool*          Find(cTag tag, cString &name) /* for(..) toolname = _Tools[i]->Name() */;
KTool*          Find(cTag tag, cV3   &ijk)    /* for(..) toolname = _Tools[i]->Name() */;

//-----
};

#endif

```

2. \toolchanger\tool.h

```

// Tool.h: interface for the Tool class.

#ifndef AFX_TOOL_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_TOOL_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "String.h"
#include "GoToParams.h"
#include "PtMeasPars.h"
#include "V3.h"
#include "Axis.h"

//-----

class KTool {
    friend class ToolChanger;

    String           _Name;
    i4               _Type;

//-----

    GoToPars*         _GoToPar;
    GoToPars*         _ABCGoToPar;

//-----

    PtMeasPars*       _PtMeasPar;
    PtMeasPars*       _ABCPtMeasPar;

//-----

    String           _QualificationArtifact;
    i4               _QualificationState;
    DateTime         _LastQualificationDate;
    ui               _MethodsSupported;

//-----

public:
    KTool(cString &name);
    virtual ~KTool();

//-----

    String           Name()                               {return _Name;}

//-----

    GoToPars*         GoToPar () const {return _GoToPar;}
    GoToPars*         ABCGoToPar() const {return _ABCGoToPar;}

//-----

    PtMeasPars*       PtMeasPar() const {return _PtMeasPar;}
    PtMeasPars*       ABCPtMeasPar() const {return _ABCPtMeasPar;}

//-----

    bool             CanDoGoTo ();
    bool             CanDoPtMeas();

//-----

    ie              Qualify(cTag tag);

```

```

//-----
virtual ie Align (cTag tag, cV3 &ijk)
{return ErrorBadContext;}
virtual ie AlignTool (cTag tag, cV3 &ijk, cr8 alpha)
{return ErrorBadContext;}
virtual ie AlignTool (cTag tag, cV3 &ijk, cV3 &uvw, cr8 alpha, cr8 beta) {return ErrorBadContext;}

//-----
virtual void EnumProp (cTag tag, ...);
virtual void GetProp (cTag tag, ...);
virtual void GetPropE(cTag tag, ...);
virtual void SetProp (cTag tag, ...);

//-----
protected:
virtual r8 A() {return 0;}
virtual r8 B() {return 0;}
virtual r8 C() {return 0;}

virtual ie A(cr8 a) {return ErrorSuccess;}
virtual ie B(cr8 b) {return ErrorSuccess;}
virtual ie C(cr8 c) {return ErrorSuccess;}

//-----
};

#endif

```

3. \toolchanger\toolab.h

```

// ToolAB.h: interface for the ToolAB class.

#ifndef _AFX_TOOLAB_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define _AFX_TOOLAB_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "Tool.h"

//-----

class ToolAB : public KTool {

Axis _AAxis;
Axis _BAxis;

//-----

public: ToolAB(cString &name);
virtual ~ToolAB();

//-----

ie Align (cTag tag, cV3 &ijk);
void EnumProp(cTag tag);

//-----

r8 A();
r8 B();

ie A(cr8 a);
ie B(cr8 b);

//-----

};

#endif

```

4. \toolchanger\toolabc.h

```
// ToolABC.h: interface for the ToolABC class.
```

```

#ifndef AFX_ToolABC_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_ToolABC_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "ToolAB.h"

//-----
class ToolABC : public ToolAB {

Axis _CAxis;
//-----

public: ToolABC(const String &name);
virtual ~ToolABC();

//-----

ie Align (cTag tag, cV3 &ijk);
void EnumProp(cTag tag);

//-----

r8 C();
ie C(cr8 c);

//-----
};

#endif

```

5. \toolchanger\gotoparams.h

```

// GoToPars.h: interface for the GoToPars class.

#ifndef AFX_GoToPars_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_GoToPars_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "../lib/String.h"
#include "Param.h"

//-----

class GoToPars {

Param _Speed;
Param _Accel;

//-----

public: GoToPars();
virtual ~GoToPars();

//-----

r8 MinSpeed() const {return _Speed.Min();}
r8 Speed () const {return _Speed.Val();}
r8 MaxSpeed() const {return _Speed.Max();}
bool CanChangeSpeed() const {return _Speed.CanChange();}
ie Speed(cr8 s) {return _Speed.Val(s);}

//-----

r8 MinAccel() const {return _Accel.Min();}
r8 Accel () const {return _Accel.Val();}
r8 MaxAccel() const {return _Accel.Max();}
bool CanChangeAccel() const {return _Accel.CanChange();}
ie Accel(cr8 s) {return _Accel.Val(s);}

//-----

void EnumProp();

//-----
};

#endif

```

6. \toolchanger\ptmeaspars.h

```
// PtMeasPars.h: interface for the PtMeasPars class.

#ifndef AFX_PtMeasPars_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_PtMeasPars_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "../lib/String.h"
#include "../lib/DateTime.h"
#include "GoToParams.h"

//-----

class PtMeasPars {

    Param      _Approach;
    r8         _Search;
    r8         _Retract;
    GoToPars   _Move;

//-----

public:      PtMeasPars();
virtual ~PtMeasPars();

//-----

    r8          MinSpeed()           {return _Move.MinSpeed();}
    r8          Speed ()            {return _Move.Speed();}
    r8          MaxSpeed()          {return _Move.MaxSpeed();}
    ie          Speed (cr8 s)       {return _Move.Speed(s);}

//-----

    r8          MinAccel()          {return _Move.MinAccel();}
    r8          Accel ()            {return _Move.Accel();}
    r8          MaxAccel()          {return _Move.MaxAccel();}
    ie          Accel (cr8 s)       {return _Move.Accel(s);}

//-----

    void        EnumProp();

//-----
};

#endif
```

7. \toolchanger\param.h

```
// Param.h: interface for the Param class.

#ifndef AFX_Param_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_
#define AFX_Param_H_79AF9D2B_A7BC_4F04_923D_1452AF559CC1_INCLUDED_

#include "IppTypeDef.h"
#include <IppErrorCodes.h>

r8          min(cr8 a, cr8 b);
r8          max(cr8 a, cr8 b);

//-----

class Param {

    r8          _Min;
    r8          _Val;
    r8          _Max;
    bool        _CanChange;

//-----

public:      Param()  {_Min=-10000; _Val=0; _Max=10000; _CanChange=Tr;}
virtual ~Param();

//-----
```

```

r8          Min    ()      const  {return _Min;}
r8          Val    ()      const  {return _Val;}
r8          Max    ()      const  {return _Max;}
bool        CanChange()   const  {return _CanChange || (_Max-_Min)<=0;}
void        EnumProp();
```

```

ie          Val(cr8 v)      {
ie          errcod = ErrorSuccess;
bool        r    = CanChange();
if (r) {
    r = v >= _Min && v <= _Max;
if (r) {
    _Val=v;
else {
    _Val = min(v, _Max);
    _Val = max(_Val, _Min);
    errcod = (v < _Min) ? ErrorParamTooSmall : ErrorParamTooLarge;}}
else {
    errcod = ErrorParamCannotBeChanged;
return errcod;}
```

```

private:
void        Min        (cr8 v) { _Min=v; }
void        Max        (cr8 v) { _Max=v; }
void        CanChange(cbo v) { _CanChange=v; }
```

```

};

#endif
```

6. Most important of lib

1. \lib\axis.h

```

// Axis.h: interface for the Axis class.

#ifndef AFX_Axis_H__B3DA30C7_5415_11D3_A481_0000F87ABD00__INCLUDED_
#define AFX_Axis_H__B3DA30C7_5415_11D3_A481_0000F87ABD00__INCLUDED_

#include "IppTypeDef.h"

class Axis {

public:
    Axis();
    ~Axis(){};
```

```

    enum     AxisType {Lin=1, Rot=2};

    char          _Name[8];
    AxisType     _Type;
    r8            _MinPos;
    r8            _ActPos;
    r8            _MaxPos;
    r8            _Pitch;
    r8            _Temperature;
    bool          _IsControlled;
    bool          _IsHomed;
```

```

public:
    Type();
    MinPos();
    MaxPos();
    Pitch();
    Temperature();
```

```

static
void         EnumProp();                                // Name,
                                         c*8
                                         // Type,          i4
                                         // MinPos,        r8
                                         // MaxPos,        r8
                                         // Temperature,   r8

//-----
};

#endif

```

2. \lib\ eulerw.h

```

// EulerA.h: interface for the EulerA class.

#ifndef AFX_EulerA_H__0E096DA3_5537_11D3_84A8_0000F87ADB6B__INCLUDED_
#define AFX_EulerA_H__0E096DA3_5537_11D3_84A8_0000F87ADB6B__INCLUDED_

#include "IppTop.h"

//-----

class EulerA {

r8           _Theta;                                // Euler angel in degree
r8           _Psi;
r8           _Phi;

//-----

public:          EulerA();
                  EulerA(cr8 theta, cr8 psi, cr8 phi);
                  EulerA(cR33 &b);

virtual        ~EulerA();

r8           Tht() const {return _Theta;}
r8           Psi() const {return _Psi;}
r8           Phi() const {return _Phi;}

//-----
};

//-----
#endif

```

3. \lib\ tag.h

```

// KTag.h: interface for the KTag class.

#ifndef AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_
#define AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_

#include <IppTypeDef.h>

//-----

class Tag {

static
i4           _TagCounter;                           // static tag
counter
i4           _Tag;                                // tag

//-----

public:        Tag(ci4 i)                         {_Tag = i;}
virtual        ~Tag()                            {}

//-----

i4           Val();                               {return _Tag;}           //
i4           NewTag();                           //

create new tag *** for client use only

//-----
};


```

```
#endif
```

4. \lib\ipptypedef.h

```
// This is the global type definition file
```

```
#ifndef _IppTypeDefDefined  
#define _IppTypeDefDefined
```

```
-----
```

```
typedef     unsigned   char    uc;           // define some shortcuts  
typedef const unsigned   char    cuc;          // for type definitions
```

```
typedef     char      char    ch;  
typedef const char      char    cc;
```

```
typedef     unsigned   short   ui2;  
typedef     signed    short   i2;  
typedef const signed    short   ci2;  
typedef const unsigned   short   cui2;
```

```
typedef     signed    int     i4;  
typedef const signed    int     ci4;
```

```
typedef     signed    int     int    ie;           // error codes  
typedef const signed    int     cie;
```

```
typedef     unsigned   int     unsigned int cui;  
typedef const unsigned   int     unsigned int ui;
```

```
typedef     unsigned   int     ich;  
typedef const unsigned   int     cich;
```

```
typedef     double    double  r8;  
typedef const double    cr8;  
typedef     float     float   r4;  
typedef const float     cr4;
```

```
typedef     bool      bool    bo;  
typedef const bool      bool    cbo;
```

```
-----
```

```
#define Fa           false // define boolean shortcuts  
#define Tr           true  
#define Nil          0
```

```
-----
```

```
#endif
```

5. \lib\ippbaseclasses.h

```
// predefined classes
```

```
#ifndef _IppBaseClassesDefined  
#define _IppBaseClassesDefined
```

```
-----
```

```
class  String;           typedef const String           cString;  
                           // data base object  
class  Tag;              typedef const Tag             cTag;  
                           // data base object  
class  ETag;             typedef const ETag            cETag;  
                           // data base object  
class  GoToPars;         typedef const GoToPars        cGoToPars;  
                           // data base object  
class  PtMeasPars;       typedef const PtMeasPars      cPtMeasPars;
```

```
-----
```

```
class  V3;               typedef const V3              cV3;  
                           // data base object
```

```

class M33;           typedef const M33          cM33;
class R33;           typedef const R33          // data base object
class T33;           typedef const T33          cR33;
//-----                                     // data base object
class T33EA;         typedef const T33EA        cT33;
class EulerA;        typedef const EulerA       // data base object
//-----                                     // data base object
//-----
class Axis;          typedef const Axis         cAxis;
class Part;          typedef const Part         cPart;
//-----
enum ErrorSeverity;  typedef const ErrorSeverity cErrorSeverity;
enum ErrorCode;      typedef const ErrorCode    cErrorCode;
//-----
#endif

```